VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Optimalizace algoritmů pro směrování vozidel

Optimization of Algorithms for Vehicle Routing

2013 Radek Tomis

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Radek Tomis**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Optimalizace algoritmů pro směrování vozidel
Optimization of Algorithms for Vehicle Routing

Zásady pro vypracování:

V rámci této diplomové práce se bude student zabývat problematikou vybranými částmi frameworku připravovaného na VŠB-TU Ostrava pro vyhledávání nejvhodnějších trasy pro navigaci (routování) dopravního vozidla. Student bude řešit ve vyvíjeném frameworku oblast implementace směrovacích algoritmů za použití vlastních datových struktur. V této práci bude dále rozšířena již existující implementace směrovacích algoritmů, které byly vyvinuty na katedře v minulosti. V této práci se bude diplomant zaměřovat především na optimalizace a zpřesňování vyvinutých řešení.

Jednotlivé body práce jsou:
1. Diplomant prostuduje možnosti standardu DATEX pro přenos informací z výstupu směrovacích algoritmů.
2. Prostudování aktuálních článku z oblasti směrovacích algoritmů.
3. Implementace vlastních datových struktur pro směrovací algoritmu a jejich naplnění daty.
4. Implementace a optimalizace moderního směrovacího algoritmu a datových struktur pro rychlé hledání tras v rámci Evropy.
5. Výpočet dostupnosti s překážkami a dynamickými prvky pomocí vybraných směrovacích algoritmů.
6. Testy rychlosti vyvinutých algoritmů.

Seznam doporučené odborné literatury:

[1] Daniel Delling, Andrew V. Goldberg, Thomas Pajor and Renato F. Werneck, Customizable Route Planning,http://research.microsoft.com/pubs/145688/crp-sea.pdf

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2013 .....

Rád bych poděkoval všem, kteří mně podporovali při studiu. Dále bych rád poděkoval vedoucímu této diplomové práce, kterým je Ing. Jan Martinovič, Ph.D., za cenné rady a připomínky při psaní této práce.

Abstrakt

Hledání nejlepší cesty je všední problém každého z nás. Všudypřítomná navigační zařízení nám značně ulehčují toto hledání. Nejlepší cestu nalezneme pouhým zadáním počátku a cíle naší cesty. Tato práce se zabývá směrováním v silniční síti a směrovacími algoritmy, které vyhledávají cesty dle našich požadavků. Ukážeme různé varianty Dijkstrova algoritmu a jejich možné optimalizace, popíšeme moderní směrovací algoritmus založený na metodě oddělovačů a provedeme experimenty s těmito algoritmy. Důraz je u směrovacích algoritmů kladen na rychlost a přesnost vyhledávání. Významná část této práce je zpracování mapových podkladů do datových struktur, bez kterých by směrování nemohlo fungovat. Na základě této práce vzniká mobilní aplikace pro výpočet dopravní dostupnosti. Aplikace je prakticky využitelná například pro hasiče, pro které byla původně i navržena.

Klíčová slova: graf, směrování, směrovací algoritmus, nejkratší cesta, dostupnost

Abstract

Searching for the best path is a common problem of all of us. Ubiquitous navigation devices are making this searching easier. Best path can be found by simply typing the start and goal of our journey. This work is about routing in the road network and routing algorithms that search paths according to our requirements. We will show different variations of Dijkstra's algorithm and their possible optimizations, we will describe advanced routing algorithm based on the separators and perform experiments with these algorithms. Emphasis is put on the speed and accuracy of routing algorithms. Processing of spatial information into our data structures is a significant portion of this work. There is a mobile application based on this work, which is computing travel accessibility. The application is practically used by firefighters, for which it was originally designed as well.

Keywords: graph, routing, routing algorithm, shortest path, accessibility

Seznam použitých zkratek a symbolů

GPS	–	Global Positioning System
LRO	–	Left-Right-Oscillate
OSM	–	OpenStreetMap
UML	–	Unified Modeling Language
WGS	–	World Geodetic System
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Vyhledávání cesty je běžný úkon každé GPS navigace, a přesto většina navigací nalezne lehce rozdílnou cestu, protože záleží na použitém směrovacím algoritmu, použitých optimalizacích a mapových podkladech. V této práci se budeme zabývat směrovacími algoritmy pro vyhledávání nejkratší cesty v silniční síti. Podíváme se na teoretické základy směrování, zpracování mapových podkladů a poté i na samotné směrovací algoritmy i s jejich optimalizacemi.

Tato diplomová práce je pokračování dříve napsané bakalářské [23] a diplomové [17] práce a vzniká jako součást většího projektu zaměřeného na směrování dopravních vozidel. Projekt by měl dohromady poskytnout kompletní řešení pro směrování a to od počátečního zpracování dat až po koncové aplikace na mobilních zařízeních a webových stránkách. Hlavní cíl této konkrétní práce je vytvořit datové struktury pro směrování a naplnit je daty, vytvořit moderní směrovací algoritmus a algoritmus pro výpočet dostupnosti a následně vše otestovat.

První část této práce je teoretický úvod ke směrování. V sekci 2 jsou vysvětleny základní pojmy a principy směrování. Na teoretický úvod navazují směrovací algoritmy v sekci 4. Postupně jsou popsány nejběžnější směrovací algoritmy a jejich optimalizace, dále se věnujeme naší implementaci moderního směrovacího algoritmu a podíváme se také na dynamické prvky směrování, které se využijí převážně u výpočtu dopravní dostupnosti.

Popis datových struktur a zpracování dat v sekci 3 je další důležitá část této práce. Na rozdíl od dřívějších prací, na které zde navazujeme, používáme mapové podklady OpenStreetMap (dále jen OSM) [5], což sebou nese určité výhody i nevýhody proti mapovým podkladům firmy Navteq.

Testy vyvinutých algoritmů a datových struktur jsou v sekci 5. Testy nám pomohou určit správné nastavení algoritmů a odhalit silné nebo slabé stránky datových struktur.

Standard DATEX je představen v sekci 6. Pokusíme se navrhnout způsob pro předávání výstupu směrovacích algoritmů ve formátu DATEX a navrhneme model, který bude schopný pojmout nejdůležitější data.

2 Směrování

Směrování je proces vybírání vhodné trasy. Probíhá ve zjednodušeném modelu reálné dopravní sítě - grafu. U směrování vozidel používáme model silniční sítě, obecně však můžeme směrovat v libovolné síti.

Graf [4] obsahuje množinu vrcholů a hran, kde hrana spojuje dva vrcholy. Vrcholy grafu znázorňují objekty a hrany propojení mezi nimi. Grafy můžeme rozdělit na neohodnocené a ohodnocené. U směrování se budeme zabývat pouze ohodnocenými grafy, jejichž vlastností je, že každá hrana má své číselné ohodnocení. Ohodnocení představuje "délku"hrany a chápeme ho ve smyslu jak "těžké"je dostat se skrz danou hranu.

Směrování [2] probíhá mezi dvěma vrcholy, zdrojovým a cílovým, a výsledkem je cesta mezi těmito vrcholy. Cestou nazýváme posloupnost vrcholů grafu takovou, že mezi každými dvěma po sobě jdoucími vrcholy je hrana, a dále platí, že žádné dva vrcholy se neopakují.

Graf obsahuje mnoho cest, které vyhovují výše uvedené neformální definici. Abychom odlišili jednotlivé cesty, budeme hodnotit jejich kvalitu. Každé cestě přiřadíme cenu (v literatuře nazývané také *váha*), což je reálné číslo určující kvalitu dané cesty. Cenu cesty definujeme jako součet ohodnocení všech hran, které se na dané cestě vyskytují.

Základní problém směrování je hledání nejkratší cesty v grafu s jedním zdrojovým a jedním cílovým vrcholem. Nejkratší cesta v grafu je ta, která má nejnižší cenu. Hledáme tedy cestu s daným zdrojovým a cílovým vrcholem a s minimální cenou této cesty. Tento problém řeší Dijkstrův algoritmus [10], popsaný dále v sekci 4.1.

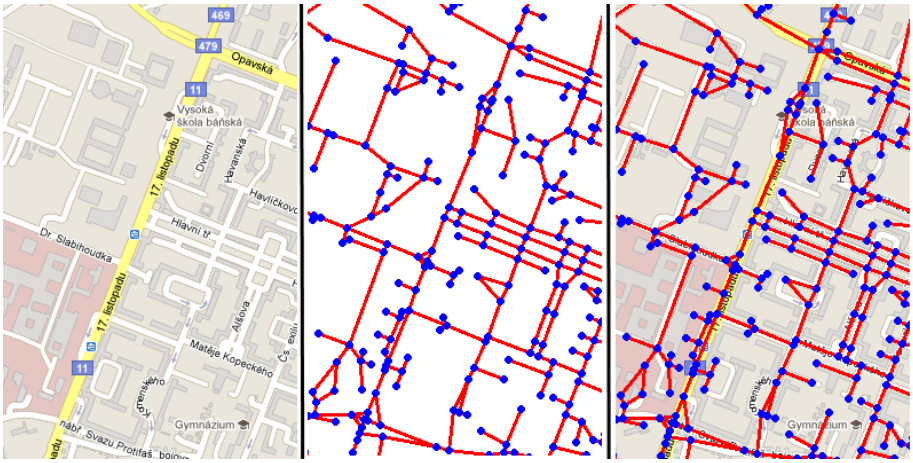
Běžně se jako ohodnocení hran používá vzdálenost mezi reálnými objekty reprezentovanými vrcholy grafu. Nejkratší cesta je u takového ohodnocení ta s nejkratší celkovou délkou. Jako ohodnocení hrany můžeme použít ale například i čas, který trvá průjezd této hrany maximální povolenou rychlostí. Nejkratší cesta je v tomto případě ta s minimálním časem potřebným k jejímu projetí. V této práci budeme pojem nejkratší cesta používat převážně ve spojení s časem. Pokud nebude řečeno jinak, budeme vždy hovořit o nejkratší cestě dle času[1].

2.1 Silniční síť

Silniční síť je forma orientovaného ohodnoceného grafu, který navíc obsahuje informace o geografickém umístění vrcholů a hran. Vrcholy představují křižovatky, zatáčky a jiné důležité objekty. Hrany jsou silnice nebo jen jejich části. Ohodnocení je nejčastěji tvořeno vzdálenostmi mezi vrcholy a je vždy nezáporné. Ukázka části silniční sítě je na obrázku 1.

Silniční síť je ze své podstaty dynamická síť. Celková struktura sítě se mění v čase například důsledkem výstavby nových silnic nebo vzniklými dopravními uzavírkami. Při směrování je však vhodné přistupovat k silniční síti jako k síti statické. Předpokládáme tedy, že se síť v čase nemění. Tento přístup nese určité výhody, jako třeba vyšší rychlost směrování a nižší nároky na implementaci algoritmů. Statické sítě mají ale samozřejmě i velkou nevýhodu, jelikož neumožňují zachytit aktuální a stále se měnící stav

[1] Neoficiální název nejkratší cesty dle času je nejrychlejší cesta.



Obrázek 1: Silniční síť – vlevo mapa oblasti, uprostřed graf silniční sítě (vrcholy modře, hrany červeně), vpravo spojení mapy a grafu.

silniční sítě. Z pohledu aktuálnosti jsou tedy statické sítě horší, než sítě dynamické. Dále v sekci 4.4 ukážeme způsob, pomocí kterého můžeme alespoň částečně "měnit"statickou síť a přidáme základní dynamické prvky.

3 Data

V této kapitole se podíváme blíže na datové struktury, se kterými pracují naše směrovací algoritmy a popíšeme postup vytvoření těchto datových struktur od počátečního zpracování mapových podkladů až po uložení do našich souborových indexů. Tento proces není jednoduchý, jak by se mohlo na první pohled zdát. Při zpracování mapových podkladů musíme řešit mnoho dílčích problémů a chyba v tomto kroku znamená nepřesné nebo zcela chybné směrování. Aktuálně používáme volně dostupné mapové podklady OpenStreetMap, na které se podíváme podrobně v sekci 3.3.

3.1 Datové struktury

Zde popíšeme způsob reprezentace našeho grafu, datové struktury a fyzické uložení samotných dat. Nejdříve se podíváme na logickou a fyzickou reprezentaci grafu, jelikož tyto reprezentace jsou důležité pro pochopení některých aspektů směrovacích algoritmů.

3.1.1 Reprezentace grafu

Graf představující silniční síť je řídký, neboli počet jeho hran je relativně malý vzhledem k počtu vrcholů, a proto reprezentujeme graf ve formě seznamu sousedů [6]. V této reprezentaci má každý vrchol odkaz na hrany, které s ním sousedí, a tyto hrany mají odkazy na vrcholy, do kterých směřují. Takové propojení je logické a je zřejmé, že umožňuje jednoduše určit u libovolného vrcholu, do jakých přilehlých vrcholů se z něj můžeme dostat a také jakou hranou. Zde musíme podotknout, že se jedná pouze o logickou reprezentaci grafu, kterou později rozšíříme o metadata.

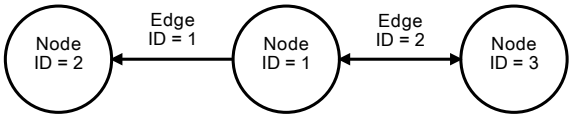
3.1.2 Logická reprezentace dat grafu

Logická reprezentace dat grafu je důležitá pro pochopení celkové struktury. Odkazy řešíme v naší reprezentaci jednoznačnými identifikačními čísly (dále jen ID) u vrcholů a hran, přičemž číslování je různé pro vrcholy i hrany. Jinak řečeno může existovat vrchol i hrana se stejným ID, dva vrcholy nebo dvě hrany se stejným ID však existovat nesmí.

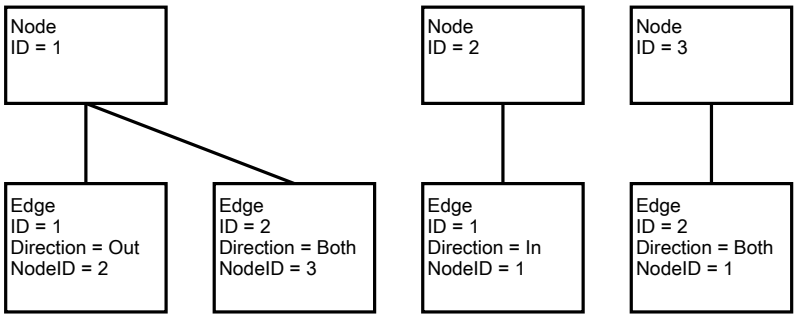
Hrany jsou v naší reprezentaci orientované, přičemž rozlišujeme jejich směr následovně:

- Vcházející (In) hrana je směrována do aktuálního vrcholu.
- Vycházející (Out) hrana je směrována od aktuálního vrcholu.
- Obousměrná (Both) hrana je kombinace vcházející a vycházející hrany. Má smysl při úspoře místa. Běžně bychom museli využít dvě hrany.

Ukázka jednoduchého grafu je pro ilustraci na obrázku 2. Směry hran určují šipky a vrcholy i hrany jsou identifikovány pomocí znázorněných ID. Graf popíšeme z pohledu vrcholu s ID 1. Ten má dvě přilehlé hrany, jednu vycházející s ID 1, která míří



Obrázek 2: Ukázka jednoduchého grafu s identifikací vrcholů i hran pomocí ID, šipky určují směry hran.



Obrázek 3: Logické uspořádání ukázkového grafu z obr. 2. Čáry znázorňují odkazy na přilehlé hrany daného vrcholu. Direction určuje směr hrany popsaný výše. NodeID je identifikační číslo cílového vrcholu, do kterého hrana směruje.

do vrcholu s ID 2 a jednu obousměrnou s ID 2, která míří do vrcholu s ID 3. Logické uspořádání tohoto grafu je na obrázku 3. Na obrázku vidíme, že odkazy na hrany jsou i u koncových vrcholů výše řečených hran (ID 2 a 3), ale mají opačný směr, pokud nejsou obousměrné, a také jiné ID cílového vrcholu. Například tedy vrchol s ID 2 má vcházející hranu s ID 1, mířící do vrcholu s ID 1.

3.1.3 Metadata grafu

Reprezentace grafu popsaná výše byla navržena pro rychlý a jednoduchý přístup k datům a přímo souvisí s fyzickou reprezentací dále v sekci 3.1.4. Nevýhoda této reprezentace je, že musíme mít každou hranu uloženu fyzicky zároveň u dvojice vrcholů. Tuto nevýhodu však převažuje jedna velká výhoda, která je v možnosti přidat metadata k vrcholům a hranám. Tyto metadata jsou uložena přímo u vrcholů a hran, a jsou velice důležitá pro správnou funkci směrovacích algoritmů. Umožňují navíc vyhledávání cest podle různých parametrů.

Vrcholy obsahují v metadatech pouze informaci o zeměpisných souřadnicích daného vrcholu ve formátu WGS 84. Metadata hran vychází z vlastností silničních segmentů, které reprezentují.

Metadata hran jsou:

- Délka segmentu silnice.
- Maximální povolená rychlost.
- Kategorie silnice.

- Přístupnost vozidel, která určuje, jaké typy vozidel mohou projet. Například osobní, nákladní nebo autobus.
- Specifické informace o daném segmentu. Například placený úsek nebo hraniční přejezd.
- Počet jízdních pruhů.

3.1.4 Fyzická reprezentace dat grafu

Fyzická reprezentace určuje, jak jsou data uložena na pevném disku. Reprezentace je téměř shodná s logickou reprezentací s přidanými metadaty. Navíc musíme zajistit efektivní uložení odkazů na hrany. Všechny zde popsané struktury mohou být používány přímo z pevného disku nebo načteny do paměti. V paměti je přístup k těmto strukturám samozřejmě rychlejší, a proto budeme převážně využívat tuto možnost.

Fyzické uložení dat

Fyzické uložení dat je realizováno grafovým souborem, což je námi definovaný binární soubor. Tento soubor je navržen tak, aby obsahoval pouze určitou část grafu, a to z důvodu optimalizace. Spojením více grafových souborů dohromady poté dostaneme původní graf. Grafový soubor nemusí být reprezentován jedním souborem na pevném disku, jak by se mohlo na první pohled zdát, viz dále sekce 3.2. Každý grafový soubor má své číslo, pomocí kterého se na něj odkazujeme. Část uchovávaného grafu bývá souvislá a většinou představuje na nějaké úrovni logický celek. Grafový soubor může být například vytvořen pro stát, kraj, město nebo část města. Ve specifických případech se může vyplatit vytvořit grafový soubor zcela jinak a to na základě algoritmů, které dělí graf na podgrafy, jako například Metis [18]. Více informací o rozdělení grafu na podgrafy uvedeme v sekci 4.3.1.

Obrázek 4 vyobrazuje jakým způsobem jsou uloženy vrcholy a hrany v grafovém souboru, a také jak fungují odkazy. Horní velký obdélník je grafový soubor označený GraphFile 0, spodní malý obdélník je ID mapa, která převádí ID vrcholů na pozice v grafovém souboru, a o které bude ještě řeč dále. Vrcholy i hrany mají konstantní velikost, a proto si můžeme dovolit uchovávat je v souboru za sebou ve formě bloků podobně, jako například v poli v paměti. Na začátku grafového souboru je uložen blok vrcholů a za ním následuje blok hran. Na obrázku jsou tyto bloky znázorněny i s indexy prvků, které jsou velice důležité pro určení pozic prvků. Písmeno X značí v obrázku aktuálně nedůležité parametry a tři tečky naznačují, že počet prvků v blocích je větší, než je skutečně nakresleno. Odkazy na hrany řešíme přidáním dvou celočíselných hodnot k metadatům vrcholu. Konkrétně se jedná o pozici hran (EdgesPosition) daného vrcholu v bloku hran (Edges) a počet těchto hran (EdgesCount). Na obrázku vidíme, že blok vrcholů (Nodes) má na indexu 0 uložen vrchol se dvěma hranami, které jsou uloženy na indexu 0 v bloku hran. První z těchto hran míří do vrcholu s ID 2. V tuto chvíli musíme určit, na které pozici v bloku vrcholů se vrchol s ID 2 nachází. Přesně pro tento účel máme připravenou strukturu nazývající se ID mapa. Popis ID mapy následuje níže. Zjednodušeně zde

řekneme, že se podíváme do ID mapy na pozici 2 a zjistíme, že daný vrchol se podle čísla grafového souboru (Node File) nachází ve stejném grafovém souboru a je na pozici 4 v bloku vrcholů (Node Index). Tímto způsobem jsme nalezli koncový vrchol první hrany původního vrcholu a mohli bychom pokračovat stejným způsobem dále.

Na závěr této části uvedeme ještě doplňující poznámku. Dříve jsme uvedli, že graf může být rozdělen na více grafových souborů. Pokud je rozdělen na více částí, pak se odkazujeme na jiné části číslem v Node File v ID mapě. ID mapa je z tohoto pohledu globální struktura obsahující umístění všech vrcholů grafu.

ID mapa

ID mapa je slovníková struktura pro převádění číselných ID na umístění v souborech. Existuje několik různých implementací, které se liší dle použití. Některé implementace šetří paměť a pracují přímo ze souboru na disku, jiné jsou kvůli rychlosti naopak v paměti. V kombinaci s grafovým souborem obsahuje ID mapa ID vrcholů a umístění je určeno jako číslo grafového souboru a index vrcholu v tomto grafovém souboru. Tím však použití ID mapy nekončí. Používá se například i pro převádění ID hran na umístění v souboru se zkratkami, viz dále.

3.2 Typy indexů

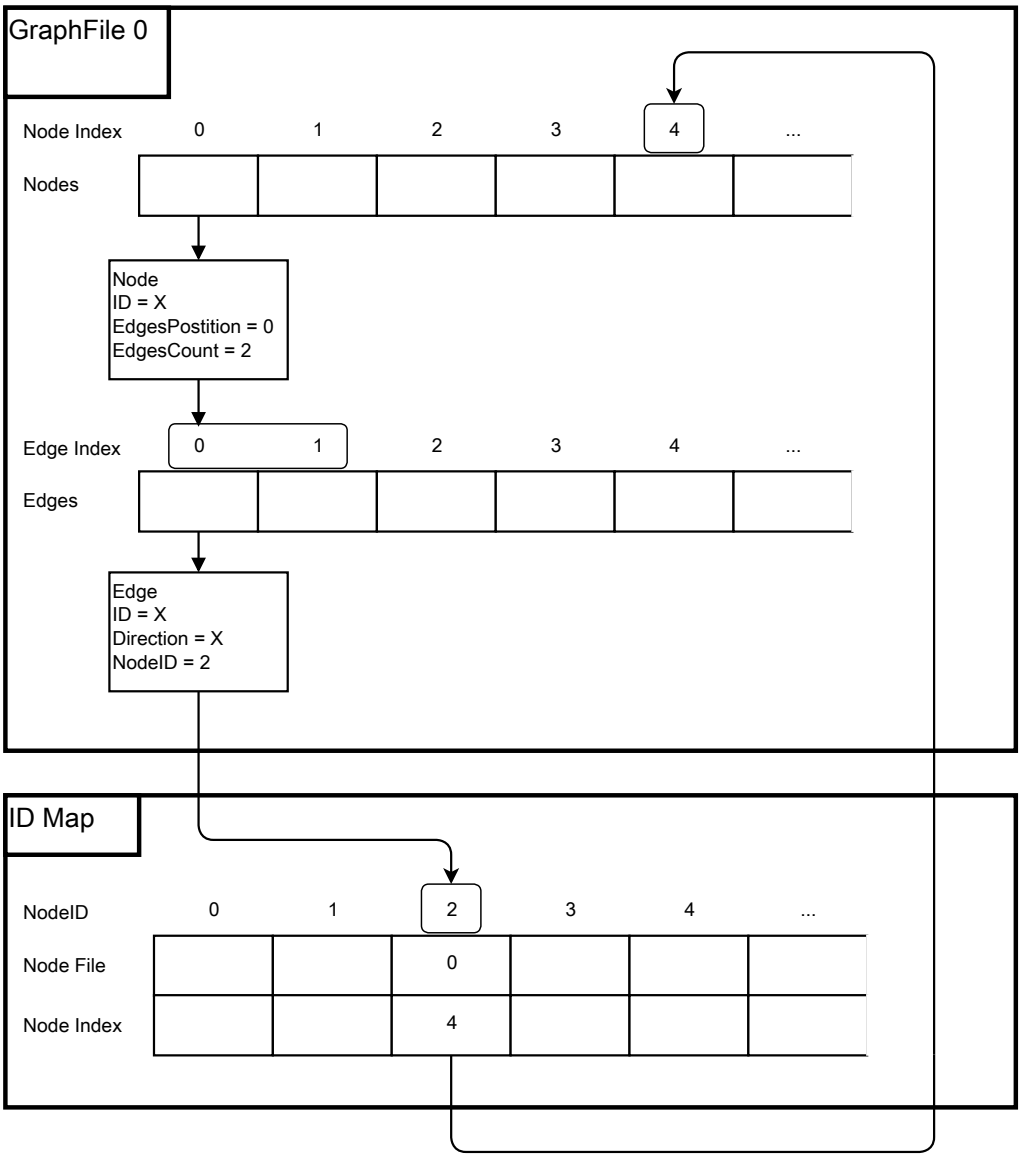
Indexem chápeme v našem pojetí souhrn prvků, které dohromady reprezentují celý náš graf. Jedná se o datovou strukturu vyšší úrovně. Index je hlavní datová struktura umožňující přístup k datům grafu. Naše směrovací algoritmy jsou uzpůsobeny k používání indexů při vyhledávání cest. Aktuálně používáme dva typy indexů. Oba typy využívají ID mapu a různě uskupené grafové soubory.

Typy indexů:

- Rozdělený index, u kterého je celý graf rozdělen na více grafových souborů, přičemž každému grafovému souboru odpovídá přesně jeden soubor na pevném disku. Tento typ indexu je výhodné použít při rozdělení grafu na menší počet logicky uspořádaných celků, například u států Evropy.

- Spojený index, ve kterém je graf stejným způsobem rozdělen na grafové soubory, ale ty jsou poté spojeny sekvenčně za sebou do jednoho velkého souboru. Výhoda je, že nezáleží na počtu grafových souborů a vždy vznikne pouze jeden soubor na pevném disku.

Tyto dva typy indexů definují pouze způsob uložení grafu na pevném disku, a ačkoliv mohou být indexy používány pro směrování přímo z pevného disku, oba typy indexů mají i své varianty, které nakopírují obsah z disku do paměti a fungují poté rychleji. Z hlediska rychlosti přístupu k datům jsou obě varianty v paměti srovnatelně rychlé. Totéž ale nemůžeme říct o variantách fungujících přímo z pevného disku. Rozdělený index zde má nepatrnou výhodu, protože ho tvoří mnoho souborů a operační systém využívá pro každý z nich malou vyrovnávací paměť, která může rychlost indexu zvýšit.



Obrázek 4: Ukázka fyzické reprezentace grafu

Zkratky

Zkratky jsou speciální datové struktury, které uchovávají data o "zkrácených"hranách. Zkrácená hrana je hrana, která byla vytvořena, aby zkrátila cestu mezi svými přilehlými vrcholy. Toho například docílíme spojením více na sebe navazujících hran do jedné jediné. Zkratky se vždy vážou k určité hraně.

V naší struktuře existují dva typy zkratek:

- Zkratky určené geometriemi, přičemž jednotlivé zkratky představují geometrie hran.
- Zkratky určené posloupností silničních segmentů, u kterých jednotlivé zkratky představují posloupnosti silničních segmentů vztahujících se k hranám.

Zkratky se většinou využívají až po skončení směrování při rekonstruování výsledné cesty. Důvod pro použití této struktury bude osvětlen dále v sekcích 3.3.3 a 4.3.1. Zatím zde můžeme pouze konstatovat, že zkratky snižují počet hran, které musí směrovací algoritmus při hledání cesty prohledat, a tím je tento algoritmus rychlejší.

3.3 Mapové podklady OpenStreetMap

OpenStreetMap je celosvětový projekt, který má za cíl vytvářet a distribuovat volně dostupná geografická data. Projekt byl založen v roce 2004 a postupně se stal velice populární díky otevřené licenci *Open Database License*. Hlavní koncept OSM spočívá ve sběru dat z volně dostupných zdrojů s kompatibilní otevřenou licencí, jako jsou například záznamy z GPS zařízení, letecké snímky a jiné digitalizované mapy.

Velká výhoda OSM je v otevřeném přístupu k upravování samotných map. Každý může vkládat nová data nebo upravovat stávající. Celkově přispělo do projektu svými daty už více než milion uživatelů a mapa je v současné době na velice dobré úrovni. Svou kvalitou však OSM zatím nepřekoná komerční mapy, a to právě díky své otevřenosti a prakticky neexistujícím standardům pro vkládání a značení dat.

3.3.1 Formát OSM

Primární formát OSM představuje XML obsahující vektorová data. Všechna data jsou uložena v hlavním XML souboru s názvem *Planet.osm* umístěném na internetové stránce projektu[2]. Tento soubor obsahuje geografická data celé zeměkoule a v současné době je jeho velikost větší než 250 GB. Dobrovolníci z řad uživatelů však nabízí i extrakt[3] dat na úrovni kontinentu, státu, kraje, města nebo dokonce jen části měst, což je pro nás výhodné, protože nebudeme muset zpracovávat množství dat větší, než je nutné.

Formát XML byl u OSM zvolen z důvodu čitelnosti dat. OSM obsahuje tři primitivní datové typy, které společně definují různé druhy geografických dat. Primitivní datové typy mají společné předepsané atributy popsané v tabulce 1 a přidávají k nim své specifické atributy.

[2] http://planet.openstreetmap.org/planet/planet-latest.osm.bz2
[3] http://wiki.openstreetmap.org/wiki/Planet.osm

Název v OSM	Český název	Popis
ID	Identifikační číslo	Identifikační číslo jedinečné v rámci celého OSM. V případě odstranění se nikdy stejné ID nevyužívá znovu.
User	Jméno uživatele	Jméno uživatele, který primitivní datový typ přidal nebo naposledy upravil.
Uid	ID uživatele	Identifikační číslo uživatele jedinečné v rámci OSM.
Visiible	Viditelnost	Vlastnost určující zda je nebo není primitivní typ viditelný.
Changeset	Množina změn	Uplatní se, pokud uživatel provede mnoho změn za krátký čas.
Timestamp	Čas	Čas poslední změny.
Version	Verze	Určuje verzi. Při každé úpravě se zvýší o jedna.

Tabulka 1: Společné atributy primitivních datových typů OSM.

Primitivní datové typy jsou:

- Uzel (node) představuje jeden bod v prostoru. Většinou se jedná o bod zájmu. Obsahuje geografické souřadnice v souřadnicovém systému WGS 84 a volitelně nadmořskou výšku.
- Cesta (way) je setříděný seznam 2 až 2000 uzlů[4]. Pokud je první a poslední uzel stejný, pak se jedná o polygon vymezující plochu, v opačném případě je to křivka. Aby nedocházelo k nedorozuměním v názvosloví, budeme většinou používat pro tento primitivní typ název OSM cesta.
- Relace (relation) se skládá ze setříděného seznamu uzlů, cest a někdy i jiných relací. Určuje logický nebo geografický vztah mezi těmito prvky.

Každý primitivní datový typ může obsahovat značky (tagy). OSM využívá pro značky systém klíčů a hodnot. Jedná se o jednoduchý systém přidávání libovolných dat k primitivním typům. Z povahy XML jsou klíče i hodnoty textové řetězce, které můžou teoreticky obsahovat cokoliv. Při směrování nás budou převážně zajímat značky s klíčem "highway", protože právě ty tvoří silniční síť. Blíže se na značky podíváme v sekci 3.3.2.

Struktura XML je ve zjednodušené formě vidět ve výpisu 1. Jedná se o ilustrativní zkrácenou verzi, kde tři tečky na místě atributu zastupují skupinu běžných společných atributů vypsaných výše. XML začíná standardním popisem verze a použitého kódování. Následuje kořenový element *osm* a verze OSM api, která určuje jaké prvky mohou být použity. V tomto kořenovém elementu jsou umístěny všechny ostatní prvky jako jeho potomci, konkrétně se jedná o blok uzlů, za ním následuje blok cest a nakonec blok relací.

OSM předepisuje mnoho klíčů a jejich možných hodnot, nicméně s vývojem OSM se tyto klíče a hodnoty mění, vznikají nové značky nebo zanikají staré a celý systém přináší

[4]Počet je definován v OSM.

mnoho problémů se zpracováním dat. Je také pouze na uživatelích, jestli budou dodržovat předpisy a vytvářet podle nich značky. Je běžné, že data obsahují mnoho nespecifikovaných značek, které si vytvářejí uživatelé pouze pro svou potřebu, čímž mohou způsobit problémy se zpracováním těchto dat ostatním uživatelům OSM. Existuje pouze předpis pro povinné prvky (primitivní typy) a vše ostatní je na úrovni doporučení pro uživatele, jak by měli data vytvářet. Neexistuje standard ani žádný jiný silnější nástroj pro validaci dat. XML schéma OSM je velice krátké a stručné a kontroluje pouze povinné atributy primitivních datových typů.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server">
  <node id="5" lat="49.8329417" lon="18.1653737" .../>
  <node id="9" lat="49.8297162" lon="18.1634586" .../>
   ...
  <node id="2" lat="49.8352613" lon="18.1665825" .../>
  <way id="8"  ... >
    <nd ref="5"/>
    <nd ref="9"/>
     ...
    <nd ref="2"/>
    <tag k="highway" v="unclassified"/>
  </way>
   ...
  <relation  id="10"  ... >
    <member type="node" ref="9"/>
     ...
    <member type="node" ref="2"/>
    <member type="way" ref="8"/>
    <tag k="type" v="route"/>
  </relation>
   ...
</osm>
```

Výpis 1: Ukázka struktury OSM XML formátu

3.3.2 Zpracování OpenStreetMap

Cíl této sekce je zjednodušeným způsobem popsat zpracování OSM do našich indexů. Samotné zpracování map se může jevit jako triviální záležitost, jelikož jde "pouze"o zpracování XML souboru, ale není tomu tak. Problémy popsané v předešlé sekci nás nutí řešit mnoho dílčích úkolů spojených s nekonzistencí OSM dat.

Hlavní myšlenka zpracování OSM je v průchodu XML, během kterého dojde k zapamatování důležitých prvků tvořících graf a následné uložení těchto dat do indexu. OSM uzly odpovídají vrcholům naší datové struktury a jejich zpracování je bezproblémové. Hrany, tak jak je máme definovány, bychom ale v OSM hledali marně. Musíme proto při zpracování provést některé důležité úpravy vstupních dat. Nejdůležitější je rozdělení OSM cest na jednotlivé segmenty, ze kterých se tyto cesty skládají. OSM cesta se může

skládat až z 2000 uzlů, z čehož jednoduchým výpočtem vyplývá, že z ní můžeme vytvořit až 1999 segmentů. Všechny segmenty jedné OSM cesty sdílí společné klíče a hodnoty. U segmentů si zapamatujeme pouze klíče a hodnoty, které nás zajímají pro směrování a po tomto kroku jsme teoreticky se zpracováním OSM hotovi. Segmenty totiž odpovídají hranám v naší datové struktuře. Dále už zbývá jen vyladit některé neduhy OSM a uložit výsledek do indexu.

Zpracování OSM cest vyžaduje převedení mnoha klíčů a hodnot do formátu metadat hran naší datové struktury. Tento úkol je obtížný, protože OSM nenutí uživatele ukládat informace k OSM cestám jednotným způsobem. Klíče mívají často nesmyslné hodnoty a kvalita OSM tímto značně trpí. Jako příklad můžeme uvést údaj o maximální rychlosti cest. OSM má v doporučení mnoho způsobů, jak tento údaj uložit a uživatelé OSM ve většině případů využívají tato doporučení, nicméně u malého počtu cest se vyskytují hodnoty v naprosto jiném formátu nebo je rychlost nelogicky zapsána slovně, případně obsahuje údaj slovní spojení, jako například "Rychlá cesta". Je jasné, že taková data se špatně zpracovávají, a i když se přizpůsobíme novým speciálním formátům jednotlivých uživatelů, tak nikdy nebudeme schopni zpracovat veškerá data. V každé novější verzi OSM dat se navíc může objevit nový speciální formát některého klíče nebo hodnoty.

OSM obsahuje i jiné anomálie, než jen výše uvedené speciální formáty. Nicméně tyto anomálie jdou ve většině případů odstranit. Anomálie můžeme rozdělit do těchto dvou skupin:

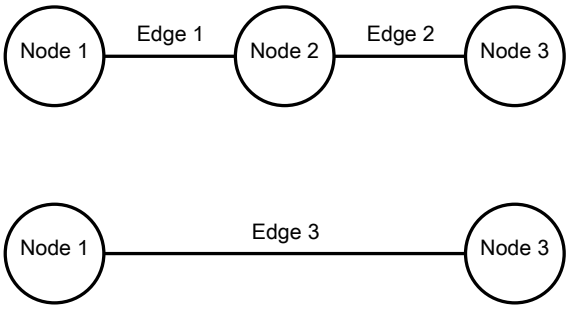
Dva uzly ve stejném umístění: v datech se objevují dva i více uzlů se stejnými GPS souřadnicemi. Pravděpodobně se jedná o chybu vzniklou při vkládání dat nebo chybu ve formátu uložených desetinných čísel, kdy jsou dva uzly příliš blízko u sebe a v datech splynou. Tuto anomálii je velice snadné odstranit, jednoduše necháme pouze jeden uzel.

Překrývající se cesty: určitá místa obsahují překrývající se OSM cesty. Překrytí vytváří více než jeden segment mezi dvojicí uzlů, přičemž každý segment může mít různé vlastnosti. Anomálie je nejvíce patrná u velkých křižovatek, u kterých existuje tři i více segmentů mezi dvojicí uzlů. Odstranění anomálie není jednoduché, jelikož musíme určitým způsobem spojit dílčí segmenty do jednoho a jejich vlastnosti nemusí být stejné.

Odstraněním anomálií jsme vyřešili zbylé problémy OSM. Výsledný index obsahuje již data, se kterými lze provádět směrování. Nicméně letmý pohled na výsledný graf odhalí ještě jeden zásadní nedostatek. Graf obsahuje velice vysoký počet nadbytečných hran, a proto musíme provést krok, který jsme nazvali minimalizace OSM grafu.

3.3.3 Minimalizace OSM grafu

Minimalizace OSM grafu je proces odstraňující nadbytečné hrany. Nadbytečné hrany vznikají při zpracování OSM dat, jelikož přistupujeme ke každému segmentu stejným způsobem a nerozlišujeme, jestli se jedná skutečně o hranu nebo pouze o geometrii nějaké jiné hrany. Geometrie je v našem pojetí tvar, který má hrana ve skutečnosti. Typický



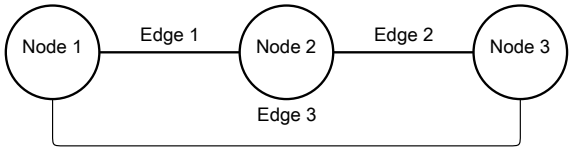
Obrázek 5: Ukázka základní myšlenky minimalizace grafu. Nahoře je ukázka části grafu vhodného pro minimalizaci, dole je výsledek po provedení minimalizace.

příklad jsou dálnice, u kterých je malý počet nájezdů a sjezdů. Graf silniční sítě pro dálnice by měl v ideálním případě obsahovat pouze vrcholy umístěné v těchto nájezdech a sjezdech a mezi nimi malý počet hran. OSM ale obsahuje mnoho silničních segmentů pro celou dálnici, čímž vznikne vysoký počet nadbytečných hran a vrcholů seřazených za sebou. Pro vyřešení tohoto problému jsme vytvořili algoritmus minimalizující graf. Dále budeme používat názvy původní graf a původní index pro dříve vytvořený graf a index a po minimalizaci vytvořený graf a index budeme nazývat minimalizovaný graf a minimalizovaný index.

Algoritmus pro minimalizaci grafu

Algoritmus pro minimalizaci grafu funguje jednoduše, musíme ale určit, kdy můžeme minimalizaci provést. Pokud jsou "za sebou"tři vrcholy propojené hranami a zároveň prostřední vrchol nemá jiné hrany, pak můžeme uvažovat o odstranění tohoto prostředního vrcholu, viz obrázek 5 nahoře. Před samotným odstraněním vrcholu musíme ještě zkontrolovat, jestli je možné spojit dvě přilehlé hrany a vytvořit z nich novou hranu se zkombinovanými vlastnostmi obou hran. Je zřejmé, že můžeme spojovat pouze hrany, jejichž směr se shoduje, případně pokud jsou obě hrany obousměrné. Dále nesmíme za žádných okolností spojit hrany, které mají rozdílnou kategorii silnice, přístupnost vozidel nebo specifické informace, protože bychom ztratili důležitá data. Zbylé vlastnosti hran můžeme spojovat, a to jako průměr například u maximální povolené rychlosti nebo sečíst například u délky hran.

Samotný algoritmus prochází postupně vrcholy původního grafu, zjišťuje, zda může aktuální vrchol odstranit a v případě, že ano, odstraníme daný vrchol a vytvoříme novou hranu mezi krajními vrcholy spojením původních hran přilehlých vrcholů. Ukázka výsledku na obrázku 5 dole. Při směrování nás ale zajímá kudy přesně cesta vede, a proto musíme zároveň s odstraňováním vrcholů a hran ukládat i původní geometrii. Geometrie určuje tvar hrany a u směrování je velice důležitá pro vizualizaci, protože díky ní neztratíme informaci o tvaru silnice. Výsledek minimalizace je minimalizovaný graf, který se stejně jako původní graf ukládá do indexu a také seznam geometrií, pomocí kterého mů-



Obrázek 6: Problém trojúhelníku. Při odstranění kteréhokoliv vrcholu bychom spojením hran museli vytvořit již druhou hranu mezi dvojicí zbylých vrcholů a vznikl by multigraf. Minimalizaci nelze provést.

žeme zpětně zrekonstruovat přesný tvar silnice. Seznam geometrií je uložen do datové struktury zkratek určených geometrií ze sekce 3.2.

Během minimalizace si musíme dát pozor na jeden speciální případ, ve kterém nesmíme provést odstranění vrcholu, i přesto, že ostatní předpoklady pro odstranění jsou splněny. Jedná se o problém trojúhelníku ilustrovaný na obrázku 6. Problém trojúhelníku by nastal, když by dva krajní vrcholy zbylé po minimalizaci již měly mezi sebou hranu[5]. V takovém případě nemůžeme vrchol odstranit, jelikož by spojením hran vznikla druhá hrana mezi stejnou dvojicí vrcholů, neboli z našeho grafu by se stal multigraf. Směrovací algoritmy neumí pracovat s multigrafy, tudíž je tato situace nepřípustná.

Počet vrcholů a hran grafu ČR vytvořeného z OSM před minimalizací a po ní je uveden v tabulce 2. Graf se zmenšil přibližně na 30% své původní velikosti. Pro srovnání jsou v tabulce 3 uvedeny i velikosti indexů pro OSM a původní Navteq. Jedná se pouze o orientační porovnání, jelikož datum vytvoření indexů a samotný typ indexů je u OSM i Navteq různý.

	Počet vrcholů	Počet hran
Původní graf	3 288 021	3 438 734
Minimalizovaný graf	970 666	1 124 244
Poměr $\frac{\text{minimalizovaný}}{\text{původní}}$	0,295	0,327

Tabulka 2: Počet vrcholů a hran v datech OSM pro graf České republiky před a po minimalizaci.

	Velikost indexu [MB]	
	Česká republika	Evropa
OSM původní graf	215,0	6 769
OSM minimalizovaný graf	64,0	2 015
Navteq	81,3	-

Tabulka 3: Velikosti indexů pro OSM a Navteq.

[5]Tuto situaci si můžeme představit na grafu vypadajícím jako trojúhelník.

4 Směrovací algoritmy

Stručně si představíme nejběžnější směrovací algoritmy, jejich optimalizované varianty a také jeden moderní směrovací algoritmus založený na metodě oddělovačů. Dále se podíváme na možnost využití dynamických prvků při směrování a výpočet dostupnosti.

4.1 Dijkstrův algoritmus

Dijkstrův algoritmus [10] je grafový algoritmus sloužící k nalezení nejkratší cesty s jedním zdrojovým vrcholem. Algoritmus si pro každý vrchol v grafu pamatuje cestu, kterou se do něj lze ze zdrojového vrcholu dostat a její cenu (délku). Algoritmus projde každý vrchol maximálně jednou a díky tomu je konečný. Pokud nás zajímá pouze cesta mezi dvěma vrcholy, můžeme prohledávání grafu ukončit, když narazíme na cílový vrchol.

Dijkstrův algoritmus hledání nejkratší cesty můžeme popsat následujícími body:

1. Počáteční vzdálenosti všech vrcholů jsou rovny nekonečnu a vzdálenost zdrojového vrcholu (od sebe sama) je 0.

2. Označíme všechny vrcholy grafu jako nenavštívené a zdrojový vrchol jako aktuální.

3. Z aktuálního vrcholu grafu vypočítáme vzdálenosti do všech přilehlých vrcholů. Pokud je takto vypočtená vzdálenost pro některý přilehlý vrchol menší než vzdálenost, kterou si pro tento přilehlý vrchol algoritmus pamatuje, přepíšeme ji. Tímto způsobem si algoritmus postupně zaznamenává pro každý vrchol grafu aktuální minimální vzdálenost od zdrojového vrcholu. Aktuální vzdálenost se v tomto kroku mění a není tedy definitivní.

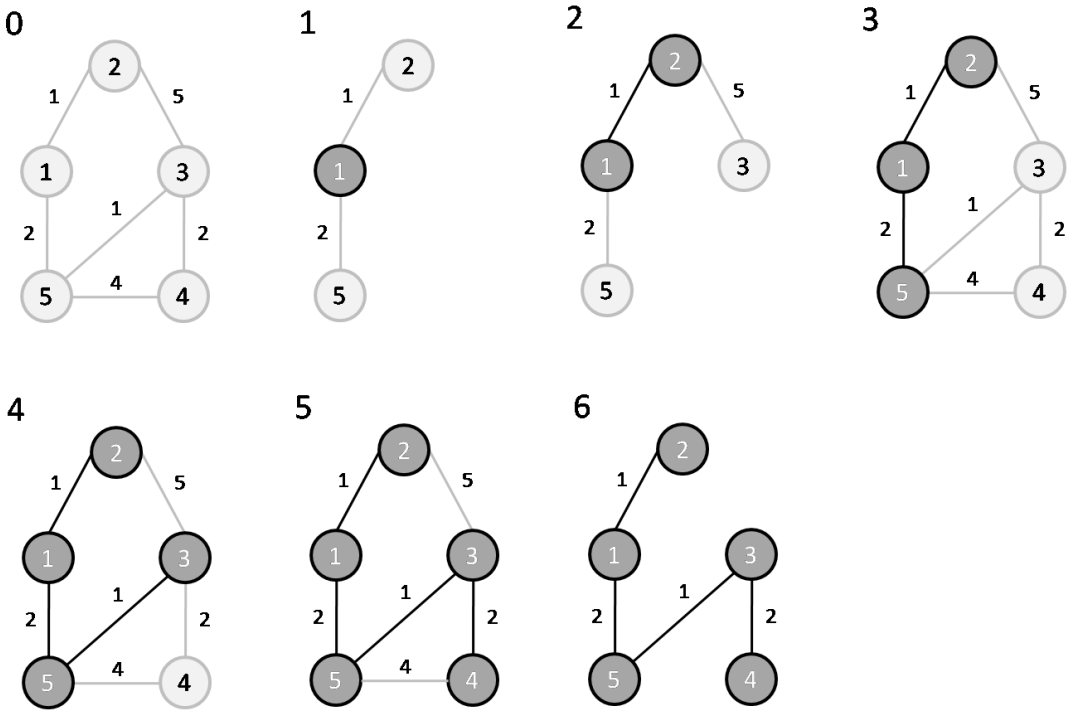
4. Aktuální vrchol označíme jako navštívený. Jeho vzdálenost od zdrojového vrcholu je již definitivní, tento vrchol už nemusíme příště kontrolovat.

5. Pokud jsme navštívili všechny vrcholy grafu, skončíme. V opačném případě vybereme z nenavštívených vrcholů ten s nejmenší aktuální vzdáleností, označíme ho jako aktuální a pokračujeme od kroku 3.

Postup algoritmu ilustruje obrázek 7. Hledáme všechny nejkratší cesty z vrcholu s číslem 1. Světle šedě jsou značeny nenavštívené vrcholy, tmavě šedě navštívené. Vrcholy, které nejsou v jednotlivých fázích algoritmu zobrazeny nebyly ani navštíveny ani nepatří mezi přilehlé vrcholy k již navštíveným vrcholům.

Dijkstrův algoritmus poskytuje pro úlohu nejkratší cesty v grafu optimální řešení. Pro danou dvojici vrcholů garantuje, že výsledná trasa je minimální a v grafu nenalezneme kratší. Daní za optimalitu je nutnost projít velké množství vrcholů, a proto je hledání velice neefektivní.

Efektivní implementace Dijkstrova algoritmu vyžaduje rychlou datovou strukturu pro ukládání dosud nenavštívených vrcholů grafu. Časová složitost algoritmu je závislá na použité datové struktuře. Množinu dosud nenavštívených vrcholů většinou reprezentujeme prioritní frontou [24]. V případě nejjednodušší implementace použijeme



Obrázek 7: Postup Dijkstrova algoritmu

jako prioritní frontu pole. Složitost vyhledání vrcholu s nejmenší aktuální vzdáleností, má v tomto případě lineární časovou složitost tj. počet operací nutných pro nalezení vhodného vrcholu je lineární funkcí počtu vrcholů v množině nenavštívených vrcholů. V tomto případě je asymptotická časová složitost $O(|V|^2 + |E|)$, kde $|V|$ je počet vrcholů a $|E|$ počet hran prohledávaného grafu. Časovou složitost je možné vylepšit použitím efektivnější prioritní fronty. Naše implementace využívá binární haldu [1], která snižuje složitost na $O((|V| + |E|)log(|V|))$.

4.2 Optimalizace směrovacích algoritmů

Optimalizace směrovacích algoritmů zajišťují efektivnější běh algoritmů a rychlejší výpočet nejkratší cesty. Rychlost algoritmů je přímo závislá na počtu prohledaných vrcholů. Snížení počtu prohledaných vrcholů je proto hlavní způsob optimalizace směrovacích algoritmů.

4.2.1 A* algoritmus

A* algoritmus [16] je vylepšená verze Dijkstrova algoritmu. Jedná se o heuristický směrovací algoritmus, který si pro každý vrchol grafu pamatuje nejen délku nejkratší cesty, kterou se do něj lze ze zdrojového vrcholu dostat, ale i odhad zbývající cesty. Odhad je nejčastěji tvořen vzdáleností k cílovému vrcholu vzdušnou čarou.

Pokud hledáme nejrychlejší cestu, to jest zajímá nás cesta s minimální jízdní dobou, musíme určit předpokládanou cestovní rychlost, se kterou vydělíme vzdálenost vzdušnou čarou a dostaneme tak odhad doby cesty k cíli. Zvolená rychlost nesmí být příliš malá, protože by nadhodnotila odhad a algoritmus by vynechal některé možné cesty. Výběr příliš vysokého odhadu rychlosti by naopak podhodnotilo ohodnocení a procházelo by se zbytečně mnoho nepravděpodobných cest.

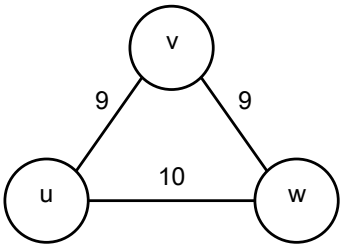
Správné určení heuristické funkce má velký vliv na kvalitu nalezené cesty. Algoritmus nejdříve prochází směr, který nejpravděpodobněji vede k cíli a zároveň bere do úvahy již uraženou vzdálenost. Optimálního výsledku [8] můžeme docílit pouze pokud použijeme přípustnou[6] heuristickou funkci. Přípustná heuristická funkce nesmí nikdy nadhodnotit ohodnocení cesty k cíli, neboli odhad ohodnocení musí být vždy nižší nebo roven než je skutečné ohodnocení cesty k cíli. Pouze tímto způsobem máme zaručeno, že nalezneme optimální řešení.

Množství prohledaných vrcholů je proti Dijkstrovu algoritmu podstatně menší. Proto tento algoritmus vyniká vyšší rychlostí výpočtu při přijatelné ztrátě přesnosti.

4.2.2 Obousměrné vyhledávání

Obousměrné vyhledávání [22] je nejběžnější optimalizační metoda směrovacích algoritmů. Princip metody je v rozdělení na dva samostatná hledání. Jedno postupuje směrem od počátečního vrcholu k cílovému a nazývá se dopředné hledání a druhé postupuje od cílového vrcholu k počátečnímu a nazývá se zpětné hledání. Vyhledávání cesty

[6] Anglicky admissible heuristic.



Obrázek 8: Problém ukončení obousměrného hledání – hledáme nejkratší cestu obousměrným Dijkstrovým algoritmem z vrcholu s označením *u* do *w*. Dopředné i zpětné hledání si v prvním kroku vyberou hrany s ohodnocením 9 a hledání se setkají ve vrcholu *v*. Nalezená cesta *u*-*v*-*w* má cenu 18, existuje však cesta *u*-*w* s cenou pouze 10.

skončí, když se oba dílčí hledání setkají přibližně uprostřed. Záměrně jsme použili termín hledání, abychom poukázali na skutečnost, že je obousměrné vyhledávání možné využít s různými směrovacími algoritmy. Nejčastěji používané varianty jsou obousměrný Dijkstrův algoritmus [15] a obousměrný A* algoritmus [7]. Z názvů obou variant je zřejmé, jaký směrovací algoritmus je u nich použit pro hledání.

Obousměrné vyhledávání je možné využít dohromady s jinými optimalizačními metodami a docílit tak efektivnějšího vyhledávání. Téměř všechny moderní směrovací algoritmy využívají obousměrné vyhledávání, protože zaručuje kratší čas běhu algoritmu. Zrychlení algoritmu vychází z rozdělení na dva samostatná hledání. Suma prohledaných vrcholů obou těchto hledání je menší, než kdyby probíhalo jedno hledání od počátečního vrcholu ke koncovému, jak jsme zvyklí.

Nalezená cesta nemusí být shodná s cestou, kterou bychom nalezli bez použití obousměrného hledání. Ukázka takového případu je na obrázku 8, na kterém hledáme nejkratší cestu z vrcholu *u* do vrcholu *w* pomocí Dijkstrova algoritmu. V prvním kroku prohledá dopředné hledání vrchol *v*, jelikož má aktuálně nejnižší ohodnocení. Stejným způsobem zpětné hledání prohledá v prvním kroku vrchol *v*. Nalezená cesta *u*-*v*-*w* však očividně není s cenou 18 minimální, protože existuje cesta *u*-*w* s cenou 10.

Příčina této nepřesnosti je v pořadí prohledávaných vrcholů. Vyhledávání nemůžeme ukončit ihned po nalezení prvního vrcholu náležícího dopřednému i zpětnému hledání, protože takový vrchol nemusí ležet na nejkratší cestě. Pro ukončení vyhledávání musíme využít silnější podmínku [15]. Během hledání si budeme pamatovat délku μ aktuálně nejkratší nalezené cesty. Na začátku je $\mu = \infty$. Když prohledáváme vrchol, který jsme již jednou prohledali v opačném směru, aktualizujeme μ, pokud je délka cesty vedoucí přes prohledávaný vrchol menší, než μ. Oba směry vyhledávání si běžně uchovávají množinu nenavštívených vrcholů a vzdálenosti do těchto vrcholů. Nazvěme min_D a min_Z minimální vzdálenosti z množiny nenavštívených vrcholů v dopředném a zpětném hledání. Obousměrné vyhledávání skončí, pokud $min_D + min_Z \geq \mu$. Tento vztah je logický a zjednodušeně říká, že hledání nesmíme ukončit, dokud může existovat kratší cesta.

4.2.3 Vyhledávání omezené silničními kategoriemi

Pravděpodobně nejstarší heuristická metoda směrování je vyhledávání omezené silničními kategoriemi [20]. Silniční kategorie jsou skupiny silnic rozdělené podle své kvality[7]. Směrovací algoritmus používající omezení silničními kategoriemi má jako vstup definováno, na jaké nejméně kvalitní silniční kategorii smí prohledávat vrcholy a hrany. Směrovací algoritmus hledá cestu pouze v této kategorii nebo v kategorii vyšší kvality a vynechává z hledání vrcholy a hrany ležící na silničních kategoriích méně kvalitních. Algoritmus musíme ještě vylepšit, jinak bychom například nenalezli žádnou cestu, kdyby byl počáteční vrchol v kategorii menší kvality. Rozšíření se týká oblastí, ve kterých může algoritmus vyhledávat bez omezení silničními kategoriemi. Tyto oblasti vytvoříme do určité vzdálenosti kolem počátečního a cílového vrcholu.

Rychlost této optimalizační techniky je závislá na mapových podkladech. Pokud kategorie vyšší kvality obsahují méně vrcholů a hran než kategorie méně kvalitní, pak se vyhledávání cesty znatelně urychlí. Samozřejmě vynecháním určité části grafu ztrácíme přesnost vyhledané cesty a riskujeme, že při špatném určení vstupní silniční kategorie cestu ani nenalezneme.

4.2.4 Obousměrné omezené vyhledávání

Obousměrné omezené vyhledávání [12] funguje na podobném principu jako vyhledávání omezené silničními kategoriemi. Na rozdíl od něho pracuje algoritmus obousměrně a tedy rozdělí vyhledávání na dva samostatná hledání. Další důležitá změna je, že na vstupu není definována kvalita silniční kategorie. Algoritmus prohledává běžným způsobem postupně z obou stran vrcholy, přičemž hledání probíhá vždy pouze na hranách silniční kategorie stejné nebo lepší kvality, než je v aktuálním vrcholu. Pokud algoritmus narazí na silniční kategorii nižší kvality, prohledá určité velice malé okolí tohoto místa. Pokračuje v hledání, když v okolí najde opět silniční kategorii vyšší kvality, v opačném případě místo vynechá z hledání. Algoritmus prohledává bez omezení kategoriemi určité okolí od počátečního a koncového vrcholu, aby se zvýšila šance nalezení cesty v případě, že bude tento počátek nebo konec umístěn přímo v nejkvalitnější silniční kategorii.

Přesnost a rychlost algoritmu jsou závislé na nastavení. Okolí od počátku a konce hledání, které algoritmus prohledá bez ohledu na silniční kategorie, zvolíme jako vzdálenost $length_{start}$ od počátečního a koncového vrcholu. Dále zvolíme $length_{max}$ jako maximální vzdálenost, do které vyhledávání pokračuje při nalezení silniční kategorie horší, než je současná silniční kategorie. Abychom dosáhli tohoto cíle, musíme si také pamatovat aktuálně nejlepší silniční kategorii a aktuální vzdálenost uraženou na horší silniční kategorii u všech prohledaných vrcholů. Aktuálně nejlepší silniční kategorie se aktualizuje pouze při přechodu na lepší silniční kategorii. Vrcholy jsou prohledávány pouze pokud je aktuální vzdálenost uražená na horší silniční kategorii menší než $length_{max}$. Aktuální vzdálenost se vynuluje, když nalezneme lepší silniční kategorii.

[7] Základní rozdělení v České republice je například na dálnice, silnice, místní komunikace a účelové komunikace, seřazeno podle kvality od nejvyšší k nejnižší. Tyto skupiny se pak dále ještě mohou dělit.

Obousměrné omezené vyhledávání funguje dobře s Dijkstrovým i A* algoritmem. Správné nastavení algoritmu je zásadní. S příliš malými hodnotami $length_{max}$ a $length_{start}$ nemusíme nalézt žádnou cestu. S vysokými hodnotami bude algoritmus prohledávat velkou část grafu a vyhledávání bude pomalejší. Vliv nastavení parametrů na vyhledávání je v sekci 5.2.

4.2.5 Obousměrné hierarchické vyhledávání

Obousměrné hierarchické vyhledávání [23] je námi již dříve vytvořený směrovací algoritmus. Algoritmus funguje ve dvou krocích. První krok je předzpracování dat. V tomto kroku algoritmus spojuje silniční kategorie podle kvality do úrovní. Druhý krok je využití těchto úrovní při směrování. Vyhledávání začíná z obou stran na úrovni nejnižší kvality. V případě, že nalezneme vrchol v úrovni lepší kvality, algoritmus přejde do této úrovně a nikdy se již nemůže vrátit do úrovně horší. Vyhledávání skončí, když se hledání z obou stran setkají na nejkvalitnější úrovni přibližně uprostřed.

Algoritmus vyhledává cesty velice rychle a kvalita nalezených cest je ve většině případů velice dobrá. V malém procentu hledání je ale výsledná cesta nepřesná, protože algoritmus přejde do úrovně vyšší kvality nesprávným vrcholem. Taková cesta se liší převážně v místě počátku nebo konce hledání. Největší problém tohoto algoritmu je tedy jeho částečná nespolehlivost, která je dána jeho návrhem.

4.3 Vyhledávání pomocí metody oddělovačů

Moderní směrovací algoritmus by měl být rychlý a přesný. Navíc by měl být i spolehlivý z pohledu celkové kvality vyhledávaných cest, neboli jinak řečeno algoritmus by měl vždy nalézt "dobrou"cestu. Tyto tři vlastnosti splňují dohromady většinou jen algoritmy, které si před směrováním vytvoří pomocné datové struktury a fungují tedy ve dvou krocích. První krok je předzpracování dat a druhý krok samotné směrování. Tuto myšlenku využívá i metoda oddělovačů [19], kterou využijeme jako základ našeho směrovacího algoritmu.

Metoda oddělovačů je obecné označení pro algoritmy, které pro svou funkci využívají graf rozdělený na menší části. Oddělovač je chápán jako množina vrcholů nebo hran, jejichž odstraněním rozdělíme graf na tyto části, neboli komponenty. Oddělovače jsou z pohledu teorie grafů grafové řezy.

Náš algoritmus potřebuje pro svou činnost graf rozdělený na určitý počet částí a předem vytvořený hraniční graf [9], viz dále sekce 4.3.1 a 4.3.2. Samotný algoritmus vyhledávání pomocí metody oddělovačů popíšeme v sekci 4.3.3.

4.3.1 Rozdělení grafu

Rozdělení grafu [3] je v teorii grafů NP-úplný problém, který se zabývá tím, zda je možné graf rozdělit na menší komponenty. Při rozdělování chceme, aby jednotlivé komponenty splňovaly určité požadavky. Algoritmus rozdělující graf musí pak tyto požadavky respektovat a řídit se jimi.

Nejběžnější požadavek je, aby byl graf rozdělen na specifický počet komponent. Další běžné požadavky jsou alespoň přibližně stejná velikost vytvářených komponent a minimální počet propojujících hran mezi komponentami. Přesně tyto tři požadavky budeme vyžadovat od rozdělujícího algoritmu. Kvalitu rozdělení budeme posuzovat podle těchto parametrů a později uvidíme, proč jsou právě tyto požadavky tak důležité.

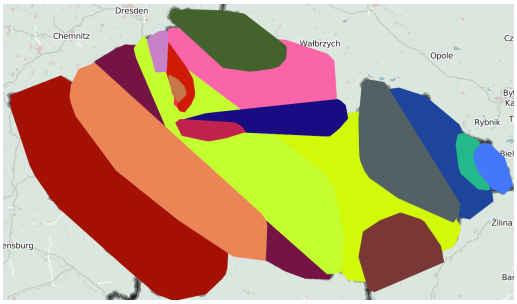
Rozdělení grafu jsme řešili dvěma způsoby. První způsob je volně dostupný software Metis [18], který se specializuje na rozdělování grafů dle různých parametrů. Druhý způsob je takzvaný "Left-Right-Oscillate"algoritmus [21] [11] (dále jen LRO), který byl již dříve vytvořen na katedře informatiky Vysoké školy báňské. Algoritmus LRO je inspirován hledáním komunit v sociálních sítích, přičemž se nesnaží rozdělovat na stejně velké části. Nesplňuje tedy jednu naši podmínku, ale může se nám hodit pro porovnání.

Na obrázku 9 můžeme vizuálně porovnat oba algoritmy. Vybrané počty komponent byly určeny algoritmem LRO. Každá komponenta má jinou barvu a barvy v jednotlivých obrázcích spolu nijak nesouvisí. Při rozdělení na malý počet částí můžeme pozorovat, že LRO rozděluje na pásy. U rozdělení na větší počet komponent vypadají výsledky obou algoritmů již velice podobně.

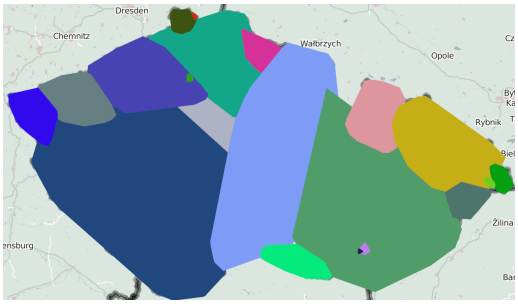
Vizuální porovnání obou způsobů rozdělení není pro naše účely postačující, a proto porovnáme výsledky obou způsobů rozdělení v číselné podobě. Pro další popis musíme vysvětlit následující dva pojmy – hraniční vrchol a hraniční hrana. Hraniční vrchol je takový vrchol, který je součástí určité komponenty a z něhož vede alespoň jedna hrana do vrcholu ležícího v jiné komponentě. Jinak řečeno je to vrchol, který leží na hranici komponenty. Hrana propojující tyto dva hraniční vrcholy se nazývá hraniční hrana. Hraniční hrana je tedy hrana ležící na rozhraní dvou komponent. Ukázka na obrázku 10. Na obrázku jsou u části naznačených komponent zobrazeny hraniční vrcholy a hraniční hrany. Komponenty samozřejmě obsahují mnohem větší počet ne-hraničních vrcholů a hran, které však nejsou v ukázce zakresleny. Záměrně také nejsou naznačeny směry hraničních hran, na kterých v této ukázce nezáleží a mohou být libovolné.

Tabulka 4 ukazuje výsledky rozdělení pro určené počty komponent. Metis 1 byl nastaven pro rozdělení s minimálním počtem hraničních vrcholů a Metis 2 pro rozdělení s minimálním součtem ohodnocení hraničních hran. Je důležité, aby celkový počet propojení mezi komponentami, neboli počet hraničních vrcholů a hran, byl minimální. Záleží nám ale také na rovnoměrném rozdělení hraničních vrcholů v komponentách, a proto jsme provedli jednoduchou statistiku, na jejímž základu vyhodnotíme kvalitu rozdělujících algoritmů. Součet ohodnocení je vypočítán jako čas potřebný pro průchod všech hraničních hran a z našeho pohledu se jedná pouze o dodatečnou informaci o daném rozdělení. Z tabulky je zřejmé, že LRO není dobré pro rozdělení na malý počet komponent, viz pásy na obrázku 9a. U rozdělení na vyšší počet komponent není kvalita LRO v počtech hraničních vrcholů o mnoho horší, než Metis 1. Směrodatná odchylka však odhalila, že LRO má mnohem větší rozmezí počtu hraničních vrcholů v komponentách, což je nežádoucí. Stejně je na tom i Metis 2, který vytváří rozdělení s příliš vysokým počtem hraničních vrcholů, a proto budeme dále používat pro rozdělení grafu pouze Metis 1, neboli variantu rozdělující na minimální počet hraničních vrcholů.

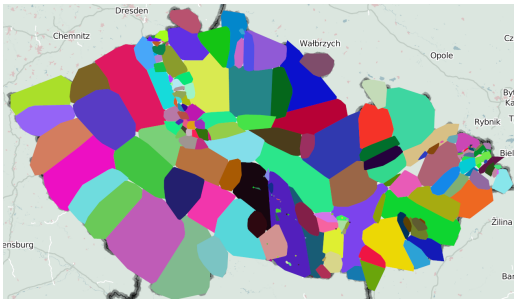
Pro zajímavost jsou na obrázcích 11a a 11b zobrazeny histogramy četností hraničních



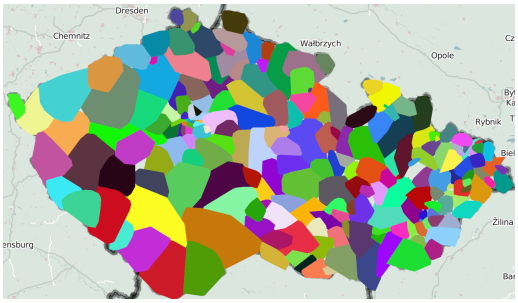
(a) LRO 19 komponent



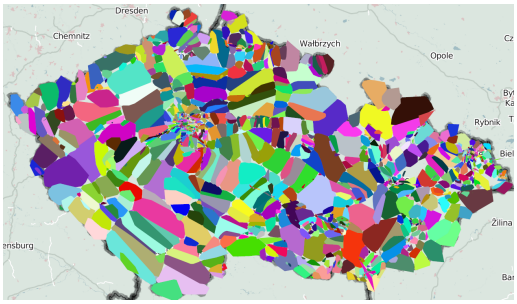
(b) Metis 19 komponent



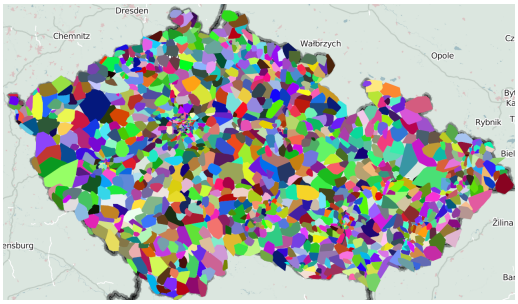
(c) LRO 216 komponent



(d) Metis 216 komponent

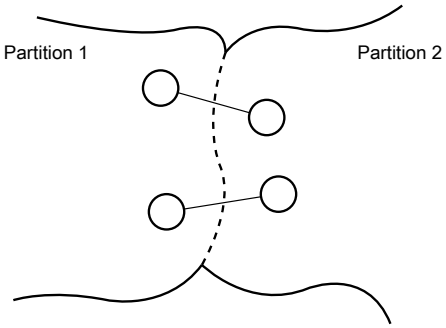


(e) LRO 1372 komponent

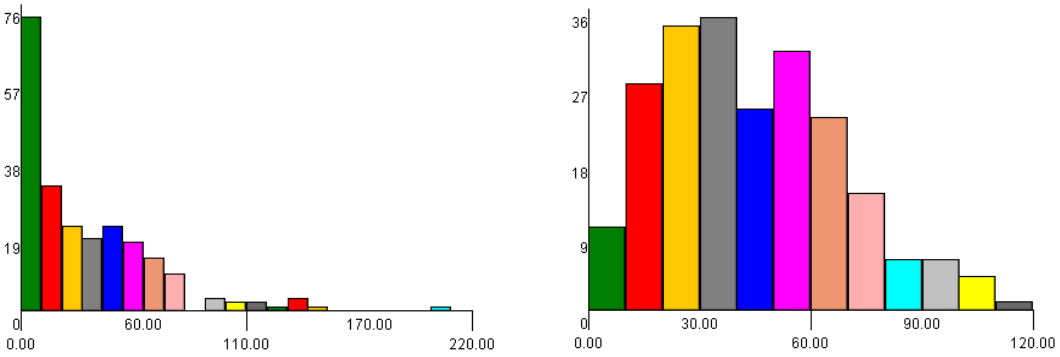


(f) Metis 1372 komponent

Obrázek 9: Ukázka rozdělení grafu pro určené počty komponent.



Obrázek 10: Ukázka hranice, hraničních vrcholů a hran. Hranice mezi naznačenými komponentami označenými čísly 1 a 2 je znázorněna čárkovaně, kružnice představují hraniční vrcholy a úsečky jsou hraniční hrany.



(a) LRO - 216 komponent

(b) Metis 1 - 216 komponent

Obrázek 11: Histogram četností hraničních vrcholů v komponentách. Na vodorovné ose jsou počty hraničních vrcholů s šířkou intervalu 10 a na svislé ose jsou samotné četnosti.

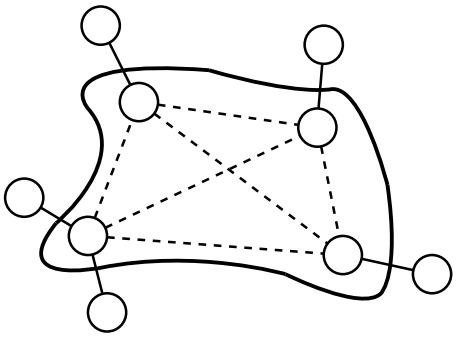
ních vrcholů v komponentách pro LRO i Metis 1 při rozdělení na 216 komponent. Histogram pro Metis 1 patří normálnímu rozdělení a četnosti jsou rozmístěny rovnoměrně. Histogram pro LRO má však charakter hyperboly a s nejvyšší pravděpodobností se jedná o Power Law distribuci.

4.3.2 Vytvoření hraničního grafu

Hraniční graf [14] je, jak již název napovídá, graf vytvořený určitým způsobem z hraničních vrcholů a hraničních hran. Základ hraničního grafu tvoří tedy hraniční vrcholy a hraniční hrany. Navíc přidáme nejkratší cesty (prakticky se jedná o nejrychlejší cesty dle časového ohodnocení) vypočítané navzájem mezi všemi dvojicemi hraničních vrcholů v každé komponentě. Část hraničního grafu znázorňující jednu komponentu je na obrázku 12.

	Hraniční vrcholy			Hraniční hrany	Součet ohodnocení [s]
	n	$\overline{x}$	σ	n	
Metis 1 - 19 k.	1026	54,0	49,3	1 070	49 727
Metis 2 - 19 k.	1523	80,1	79,4	1 642	4 911
LRO - 19 k.	3101	163,2	143,4	3 488	223 377
Metis 1 - 216 k.	6048	28,0	13,8	6 206	265 052
Metis 2 - 216 k.	9630	44,6	23,8	9 866	27 667
LRO - 216 k.	6968	32,3	33,1	7 728	481 351
Metis 1 - 1372 k.	19858	14,5	6,3	20 444	706 482
Metis 2 - 1372 k.	28197	20,6	9,0	29 250	87 914
LRO - 1372 k.	20504	14,9	13,0	22 708	1 365 077

Tabulka 4: Přehled výsledných rozdělení grafu pro vybrané počty komponent. Celkový počet hraničních vrcholů a hran ve všech komponentách značí n, $\overline{x}$ je aritmetický průměr počtu vrcholů v komponentách a σ směrodatná odchylka. Součet ohodnocení je suma času potřebného k překonání všech hraničních hran. Metis 1 je nastaven pro rozdělení s minimálním počtem hraničních hran, Metis 2 je pro minimální součet ohodnocení hraničních hran.



Obrázek 12: Ukázka části hraničního grafu. Znázorněna je jedna komponenta, kružnice představují hraniční vrcholy, úsečky kreslené plnou čarou jsou hraniční hrany a čárkované úsečky představují nejkratší cesty. Nejkratší cesty jsou vypočítány mezi všemi dvojicemi hraničních vrcholů dané komponenty.

Hraniční graf se tedy skládá z hraničních vrcholů, hraničních hran a nejkratších cest mezi hraničními vrcholy v rámci každé komponenty. Je zřejmé, že musíme určit parametry vyhledávání, se kterými se výpočet nejkratších cest provede. Tyto parametry jsou společné pro výpočty všech nejkratších cest v daném hraničním grafu a vymezují hraniční graf pro konkrétní použití. Parametry vychází převážně z informací uložených u hran popsaných v sekci 3.1.3.

Nejběžnější parametry vyhledávání jsou:

- Povolení vyhledávání na placených úsecích.
- Hledání cesty průjezdné pro určitý typ vozidel.

Výpočet nejkratších cest je v tomto kroku realizován běžným Dijkstrovým algoritmem, jelikož ten zajišťuje nalezení skutečně nejkratší cesty. Všechny cesty musíme hledat v obou směrech, neboli dopředně i zpětně. Dopředně prohledáváme pouze hrany vycházející a obousměrné a obdobně zpětně prohledáváme pouze hrany vcházející a obousměrné. Tímto způsobem zajistíme, že nalezneme i jednosměrné nejkratší cesty.

Důležitá otázka je, zda můžeme provést výpočet nejkratších cest mezi hraničními vrcholy některé komponenty omezeně – pouze v části grafu odpovídající této komponentě. Odpověď je samozřejmě ne. Představme si, že naše komponenta je centrum města. Centrum města je velice husté, co se týká počtu vrcholů a hran a má pouze několik příjezdových cest, takže i počet hraničních vrcholů je malý. Je velice nepravděpodobné, že cesta skrz toto centrum bude kratší, než libovolný silniční obchvat kolem centra. Takový obchvat je ale součástí jiné komponenty. Proto i výpočet všech nejkratších cest musíme provést na větší oblasti.

Pokud bychom chtěli mít jistotu nalezení skutečných nejkratších cest, museli bychom provést výpočet na celém vstupním grafu. Toto řešení je však časově neefektivní. Když by mezi dvojicí hraničních vrcholů neexistovala cesta[8], pak by se výpočet zastavil až po prohledání celého grafu. Z podstaty silniční sítě také nepředpokládáme, že by mohla nejkratší cesta pro některou komponentu existovat ve vzdálenosti několika set kilometrů od této komponenty. Z toho důvodu je vhodné omezit oblast vyhledávání Dijkstrova algoritmu na určité okolí daných komponent. Test v sekci 5.4 ukazuje, jakým způsobem ovlivňuje velikost tohoto okolí kvalitu hraničního grafu.

Záměrně jsme zatím neuvedli, jakým způsobem se ve výsledku uchovávají nejkratší cesty mezi hraničními vrcholy. Pravděpodobně nejjednodušší způsob je přidat tyto nejkratší cesty do hraničního grafu. Cesty se ale mohou skládat z vysokého počtu vrcholů a hran, které by značným způsobem zvětšily hraniční graf, a proto je výhodnější použít jiný způsob uchování nejkratších cest. Jelikož byly tyto cesty vyhledány se stejnými parametry, můžeme si dovolit vytvořit z každé cesty jednu hranu, říkejme ji zástupná hrana, která bude mít zkombinované vlastnosti všech hran vyskytujících se na této cestě. Touto zástupnou hranou poté nahradíme v hraničním grafu původní cestu. Princip je podobný jako u minimalizace v sekci 3.3.3. Abychom neztratili informace o samotné cestě, musíme si samozřejmě pamatovat i celou původní cestu. Všechny nejkratší cesty si uložíme

[8] Tato situace nastává běžně u jednosměrných silnic. Jelikož provádíme výpočet v obou směrech, nebude logicky jedna cesta nalezena.

do datové struktury zkratek určených posloupností silničních segmentů a pamatujeme si u nich, do jaké zástupné hrany jsme je zkombinovali. Nejsou tedy součástí hraničního grafu a můžeme podle nich zpětně zrekonstruovat zástupnou hranu na původní cestu.

Dané parametry jsou stejné v rámci jednoho hraničního grafu, a proto musíme pro každou skupinu parametrů vytvořit nový hraniční graf. Toto je nevýhoda metody oddělovačů, protože pro každé specifické použití musíme vytvořit konkrétní hraniční graf. Můžeme si to však dovolit, protože hraniční graf je co do velikosti mnohem menší, než graf původní.

Pokud jedna komponenta původního grafu obsahuje n hraničních vrcholů, pak část hraničního grafu pro tuto komponentu má n hraničních vrcholů a maximálně $n^2 - n$ zástupných hran. Aby mělo smysl metodu oddělovačů použít, musí být počet hraničních vrcholů, zástupných hran a hraničních hran nižší, než je počet vrcholů a hran v původním grafu. Je zřejmé, že počet zástupných hran je přímo závislý na počtu hraničních vrcholů. Proto snížením počtu hraničních vrcholů můžeme snížit počet zástupných hran. Pouhé snížení celkového počtu hraničních vrcholů však nestačí. Druhá mocnina ve vzorci má vliv na celkový počet zkracujících hran. Pokud budou počty hraničních vrcholů ve všech komponentách průměrně stejné, bude počet zkracujících hran malý. Při rozdílných počtech hraničních vrcholů v komponentách bude počet zkracujících hran mnohem vyšší.

Hraniční graf se po vytvoření uloží do indexu popsaného v sekci 3.2.

4.3.3 Algoritmus vyhledávání pomocí metody oddělovačů

Algoritmus vyhledávání pomocí metody oddělovačů [13] je založen na obousměrném vyhledávání s Dijkstrovým algoritmem a pracuje ve dvou fázích. Stejně jako u obousměrného vyhledávání se skládá z dopředného a zpětného hledání. V první fázi algoritmus prohledává pouze komponenty, ve kterých se nalézá počáteční a koncový vrchol. Dopředné hledání prohledává postupně celou komponentu, ve které je počáteční vrchol a zpětné hledání prohledává celou komponentu, ve které je koncový vrchol. Oba hledání si pamatují prohledané hraniční vrcholy. Jakmile jsou tyto dvě komponenty prohledány, přejde se do druhé fáze. Oba hledání pokračují ve druhé fázi ze zapamatovaných hraničních vrcholů a vyhledávají už jen na hraničním grafu. Cesta je nalezena, když se tyto dvě hledání potkají.

Hraniční graf obsahuje pouze malé množství vrcholů a hran v porovnání s grafem původním, a proto je v něm vyhledávání velice rychlé. Vysoká rychlost algoritmu se projeví převážně při hledání cest na větší vzdálenost. Jak jsme již zdůraznili dříve v sekci 4.3.1, záleží na kvalitě rozdělení grafu. Čím kvalitnější je rozdělení grafu, tím méně hraničních vrcholů hraniční graf obsahuje a tím rychlejší je samotné vyhledávání cesty.

Kvalita rozdělení grafu je pro rychlost vyhledávání zásadní, ale není to jediný faktor ovlivňující výsledky algoritmu. Velice důležitý je i počet komponent, na který graf rozdělíme. Rozdělením na malý počet komponent vytvoříme velké komponenty, tj. komponenty, které mají vysoký počet vrcholů a hran, a to bude mít za následek, že první fáze vyhledávání bude pomalá a druhá fáze rychlá. Rozdělení na velký počet komponent bude mít na algoritmus přesně opačný vliv. První fáze bude velice rychlá a druhá bude pomalejší. Toto tvrzení bylo prakticky ověřeno testem v sekci 5.3.

4.4 Dynamické prvky směrování

Jedna z klíčových součástí dnešních směrovacích algoritmů je možnost využití dynamických prvků při směrování. Běžně přistupujeme k silniční síti jako ke statickému grafu. Nicméně z povahy silniční sítě je jasné, že takový přístup nemusí vždy fungovat. Pokud se na libovolné silnici stane nehoda, měli bychom být schopni vyhledat cestu jinudy.

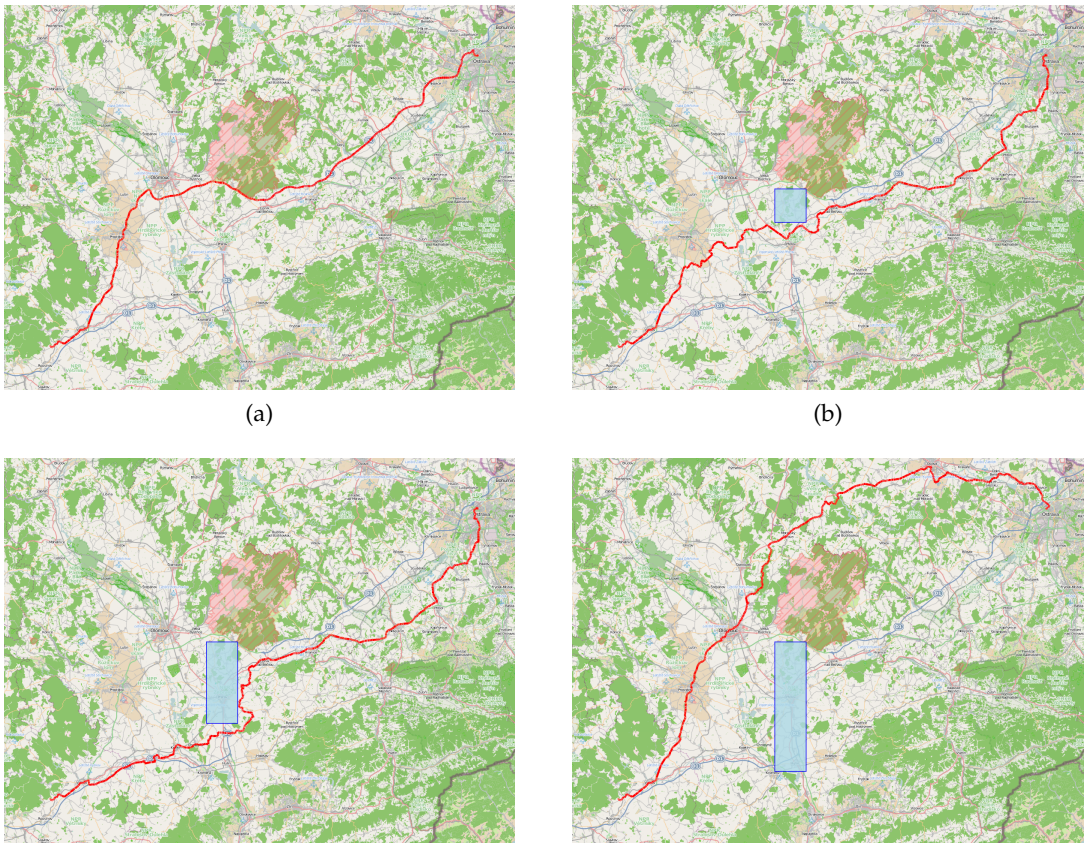
Nejčastější nestandardní události, které nastávají v silniční síti, jsou uzavírky silnic a dopravní nehody. Tyto události musíme nějakým způsobem zohlednit při směrování. Důležitou otázkou je, jak reprezentovat tyto události při směrování. Teoreticky nejjednodušší způsob by byl postupně měnit s každou nastalou událostí graf. Toto řešení je však nepraktické, jelikož se změnou grafu musíme vygenerovat nový index pro tento graf. Proto je vhodné tento způsob využít pouze pro dlouhotrvající změny silniční sítě. Většina událostí však netrvá tak dlouho, a proto jsme směrovací algoritmy rozšířili o systém restrikcí.

Restrikce představují dopravní omezení vztahující se k určité oblasti. Mohou být určeny dvěma způsoby. První z nich je spjat přímo s grafem. Vytvoříme množinu vrcholů a hran, které nejsou průjezdné a v průběhu hledání pouze kontrolujeme, aby aktuálně prohledávané vrcholy a hrany neprocházely nepatřily do této množiny. Druhý způsob určování restrikcí je podstatně rozmanitější. Pro restrikci vytvoříme geograficky určený polygon, který vymezuje neprůjezdnou oblast v prostoru. Tímto způsobem můžeme jednoduše odstranit z hledání celé oblasti, jako například zatopená místa při povodních. Každý polygon v sobě může mít jednu nebo více děr, které jsou reprezentovány taky polygony.

Implementace restrikcí určených množinou vrcholů a hran je velice jednoduchá. Jelikož všechny dříve uvedené algoritmy vychází z Dijkstrova algoritmu, můžeme si dovolit popsat obecné řešení. Kontrola restrikcí probíhá ve 3 kroku Dijkstrova algoritmu popsaného v sekci 4.1. U restrikcí určených množinou vrcholů a hran zkontrolujeme, zda aktuální vrchol nebo prohledávané hrany patří do této množiny. Pokud ano, vynecháme je z hledání.

Restrikce určené polygonem jsou o něco složitější. Musíme implementovat algoritmus detekce bodu v polygonu a také detekce protnutí úsečky a polygonu. Detekci bodu v polygonu využijeme u vrcholů, a to tak, že vyzkoušíme, zda bod určený souřadnicemi vrcholu leží v polygonu. Detekci protnutí úsečky a polygonu využijeme obdobně u hran. Z dvou vrcholů tvořících hranu vytvoříme úsečku a zkontrolujeme, zda protíná polygon. Pokud je bod v polygonu nebo úsečka protíná polygon, vynecháme příslušný vrchol nebo hranu z hledání.

Ukázka restrikce určené polygonem je na obrázku 13. Dijkstrovým algoritmem vyhledáváme cestu mezi dvěma vrcholy, přičemž tato cesta nesmí procházet skrz modře zaznačené polygony. V bodě 13a je červeně znázorněna skutečná nejkratší cesta. V dalších bodech jsou naznačeny různě velké polygony a nejkratší cesta se podle nich mění.



(a) (b)

(c) (d)

Obrázek 13: Ukázka dynamických prvků směrování. Modře jsou naznačeny restrikce určené polygonem. Nejkratší cesty, zakresleny červeně, nesmí procházet těmito polygony. Bod 13a znázorňuje skutečnou nejkratší cestu.

4.5 Dostupnost

Dostupnost je časová dosažitelnost z určitého místa nebo míst do okolí. Časová dosažitelnost je podobně jako u směrovacích algoritmů dojezdový čas, nicméně tento čas vypočítáváme pro mnohem větší oblast, přičemž nás zajímá čas dojezdu ve všech bodech, do kterých se můžeme dopravit. Algoritmus pracuje, na rozdíl od běžných směrovacích algoritmů, s množinou počátečních vrcholů a definovanou oblastí okolo těchto vrcholů. Pro tento vstup vypočítáme výstup neboli dostupnost. Oblast okolo vrcholů může být definována geograficky, jako maximální dojezdová doba nebo například maximální vzdálenost od počátečních vrcholů.

Výpočet dostupnosti probíhá s pomocí upraveného Dijkstrova algoritmu. Algoritmus najde pro každý počáteční vrchol všechny nejkratší cesty do definovaného okolí, a to pouze jedním výpočtem pro každý počáteční vrchol. Je jasné, že pokud se definované okolí počátečních vrcholů překrývají, budou se překrývat i nalezené cesty. Výstup dostupnosti je dále možné zpracovat. Nejběžnější úprava výstupu dostupnosti spočívá v ponechání pouze části cest s nejnižšími dojezdovými časy pro každý vrchol. V každém vrcholu ponecháme tedy pouze nejnižší dojezdový čas, přičemž si samozřejmě pamatujeme i cestu nebo jenom část cesty, kterou se zde dostaneme. Tato úprava je provedena pouze pro "zmenšení"výstupu a je vhodné ji provést, pokud budeme data dále zpracovávat.

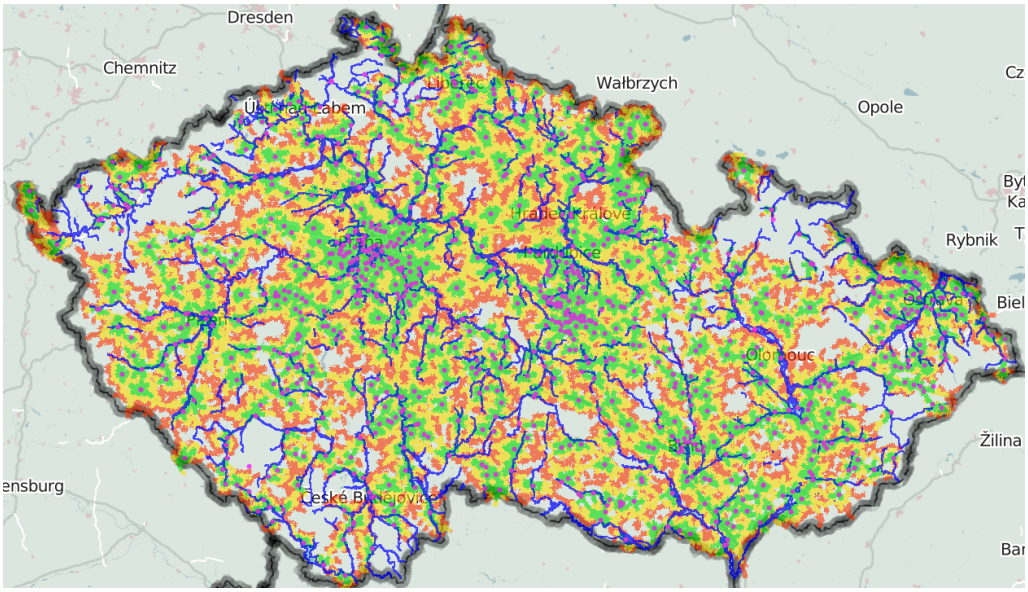
4.5.1 Dostupnost hasičských stanic

Dostupnost hasičských stanic je jedna z realizovaných aplikací algoritmu. Aplikace byla vytvořena speciálně pro hasiče a umožňuje nalezení dostupnosti z hasičských stanic. Můžeme tak optimalizovat čas příjezdu hasičských vozidel k zásahu. Navíc je možné využít dynamické prvky podobně jako v sekci 4.4.

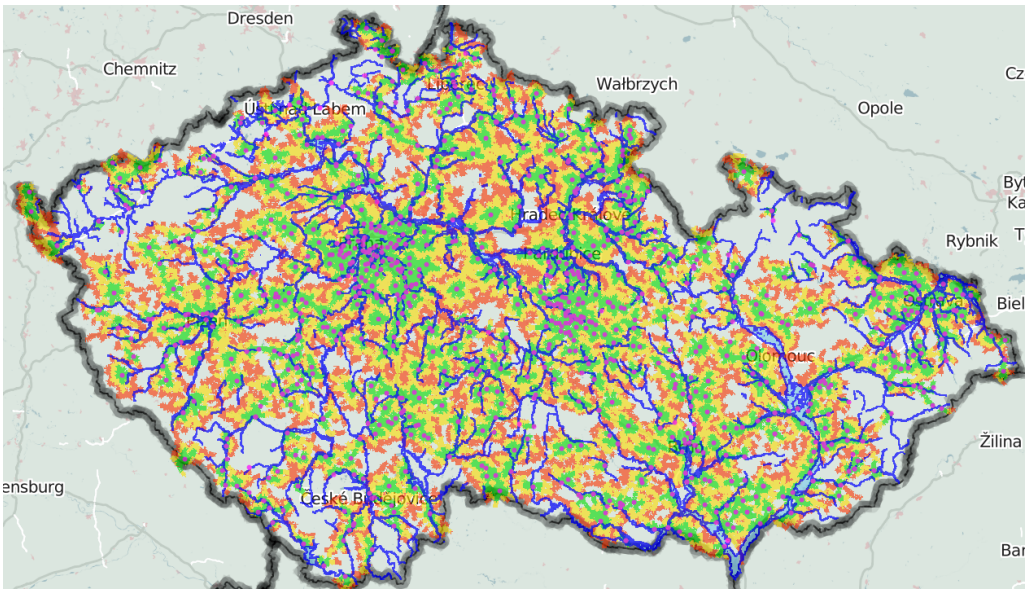
Dynamické prvky mohou simulovat krizové situace a připravit nás na ně nebo s využitím aktuálních dopravních dat můžeme během krizových situací vyhledávat nejlepší cesty v reálném čase. Jako příklad takové krizové situace můžeme uvést povodně. Pro simulaci dostupnosti při povodních jsme využili volně dostupná data [9] reálných záplavových území z předešlých let. Záplavová území jsou v datech reprezentována ve formě multipolygonů. Každé záplavové území tvoří jeden vnější polygon a mnoho vnitřních polygonů představujících díry v tomto polygonu. Převedení této struktury na polygony bez děr je velice pracné, a proto je jednodušší rozšířit dříve popsaný způsob pro použití dynamických prvků při směrování o restrikce určené multipolygonem. Jedná se o jednoduchou úpravu, jelikož vše potřebné je již definováno u restrikcí určených polygonem.

Na obrázku 14 je znázorněna dostupnost hasičských stanic při stoleté vodě. V místech, kde je dostupnost z více počátečních vrcholů zároveň, se vybere pouze ten s nejnižším časem. Mapa a k ní související data určují, ze které hasičské stanice můžeme v nejkratším čase provést výjezd do určitého místa.

[9] Data jsou dostupná na adrese http://www.dibavod.cz/index.php?id=27, pod kategorií D – záplavová území.



(a) Dostupnost hasičských stanic při pětileté vodě



(b) Dostupnost hasičských stanic při stoleté vodě

Obrázek 14: Dostupnost hasičských stanic. Nezatopené hasičské stanice jsou znázorněny fialově a záplavová území modře. Dostupnost je vykreslena dle dojezdové doby – zeleně do 5 minut, žlutě do 10 minut a červeně do 15 minut. Tam, kde není žádná z uvedených barev, je dostupnost nad 15 minut.

5 Experimenty

Experimenty se zaměřují na dříve popsané směrovací algoritmy ze sekce 4. Přechod na mapové podklady OSM má za následek snížení kvality dat a to převážně u uspořádání silničních segmentů dle kategorie. Dříve používané mapové podklady firmy Navteq dodržovaly ve velké míře schéma, ve kterém se přilehlé silnice nelišily o více než jednu silniční kategorii. OSM však žádné takové schéma nedodržuje, a proto není možné provádět testy vyhledávání omezeného silničními kategoriemi, jelikož s takovými daty nefunguje.

Experimenty proběhly na počítači s následující konfigurací – operační systém Windows 7 64 bitů, procesor Intel Core i5-750 taktovaný na 2,66 GHz a 4 GB operační paměti.

5.1 Porovnání s původní verzí

Celá tato práce je vytvořena jako pokračování dříve vytvořené bakalářské práce [23], která navazovala na ještě dříve vydanou diplomovou práci [17]. Obě tyto práce obsahovaly určité algoritmy a datové struktury, které se v mnohém podobají těm současným. Proto je vhodné srovnat původní verzi se současnou. Nové datové struktury a indexy bohužel nejsou kompatibilní s těmi starými a není tedy možné provést srovnání těchto dvou verzí spuštěním vyhledávacího testu. Z toho důvodu provedeme porovnání jinak. Z původní bakalářské práce vybereme jeden test a zrekonstruujeme ho se současnou verzí. Kvůli jednoduchosti vybereme pro porovnání test rychlosti Dijkstrova algoritmu. Současným Dijkstrovým algoritmem vyhledáme cesty mezi stejnými místy, které byly použity v původní verzi a porovnáme čas vyhledávání.

Tabulka 5 porovnává výsledky obou verzí. Původní časy byly měřeny s indexem v paměti, čas označený Současný 1 je změřen s indexem na pevném disku a Současný 2 je opět s indexem v paměti. Vidíme, že u současné verze s indexem v paměti došlo přibližně k pětinásobnému zrychlení při vyhledávání Dijkstrovým algoritmem. Příčina tohoto zrychlení je v použití novějších datových struktur a indexů a lepší optimalizace Dijkstrova algoritmu. Současný čas pro index na pevném disku se může zdát na první pohled vysoký, ale není tomu tak. Musíme si uvědomit, že Dijkstrův algoritmus prohledává obrovské množství vrcholů a pevný disk je omezen rychlostí operace seek. Přesnost výsledků s indexem na pevném disku mohla být také ovlivněna vyrovnávací pamětí disku. Následující testy jsou prováděny už pouze s indexy v paměti.

5.2 Nastavení obousměrného omezeného vyhledávání

Rychlost a přesnost obousměrného omezeného vyhledávání ze sekce 4.2.4 je závislá na nastavení parametrů $length_{start}$ a $length_{max}$. Pro každé z vybraných hodnot $length_{start}$ a $length_{max}$ bylo provedeno sto vyhledávacích testů předem vybraných dvojic vrcholů pro variantu obousměrného omezeného vyhledávání s Dijkstrovým algoritmem.

Tabulka 6 ukazuje přesnost obousměrného obousměrného Dijkstrova algoritmu. Názvy parametrů $length_{start}$ a $length_{max}$ jsou pro účely této a následující tabulky 7 zkráceny na l_s a l_m. Prodloužení času cesty je průměr ze sta testovacích cest a je bráno ve vztahu

Trasa	Čas výpočtu [ms]			Poměr $\frac{\text{Původní}}{\text{Současný 2}}$
	Původní	Současný 1	Současný 2	
Opava–Pardubice	13 034	15 053	2 559	5,1
Břeclav–Šluknov	12 963	14 712	2 591	5,0
Jihlava–Třeboň	6 713	7 244	1 372	4,9
Vysočany–Svitavy	8 265	8 898	1 683	4,9

Tabulka 5: Porovnání času výpočtu Dijkstrova algoritmu v původní a současné verzi. Původní časy byly měřeny s indexem v paměti, čas označený Současný 1 je změřen s indexem na pevném disku a Současný 2 je s indexem v paměti.

	Prodloužení času cesty [%] / Počet nalezených cest			
	$l_m = 5$ km	$l_m = 10$ km	$l_m = 15$ km	$l_m = 20$ km
$l_s = 10$ km	3,1 / 12	2,6 / 22	2,9 / 32	2,6 / 45
$l_s = 20$ km	2,6 / 29	2,6 / 41	2,4 / 54	2,1 / 63
$l_s = 30$ km	1,3 / 47	1,0 / 58	1,0 / 63	0,9 / 74
$l_s = 40$ km	0,8 / 60	1,0 / 70	1,1 / 73	0,8 / 85
$l_s = 50$ km	0,8 / 72	0,9 / 81	0,8 / 86	0,6 / 94

Tabulka 6: Přesnost obousměrného omezeného Dijkstrova algoritmu pro určené parametry $length_{start}$ a $length_{max}$. Hodnoty prodloužení času cesty a celkový počet nalezených cest jsou odděleny znakem "/".

ke skutečné nejkratší cestě vypočítané Dijkstrovým algoritmem. Počet nalezených cest je celkový počet cest z testovací množiny sta cest, které algoritmus nalezl. Hodnoty jsou kvůli úspoře místa oddělovány znakem "/". Tabulka prozrazuje, že se zvyšujícími se parametry $length_{start}$ a $length_{max}$ se prodloužení času cest snižuje a zároveň se zvyšuje počet nalezených cest. Ani nejvyšší hodnoty parametrů však nezaručily nalezení všech sta cest. Pro nalezení všech cest bylo nutné použít kombinaci parametrů $length_{start} = 50$ km a $length_{max} = 25$ km. Algoritmus tedy nevyniká po stránce kvality vyhledávaných cest, ve specifických případech je ale vhodné ho využít. Jeden takový případ je, pokud požadujeme, aby se kvalita silničních kategorií z obou stran hledání zvyšovala, přičemž nejvyšší kvalita bude uprostřed cesty. Toto je běžné pro chování řidiče, který nejprve hledá místní hlavní silnici, pak silnici první třídy, poté libovolnou rychlostní komunikaci a následně totéž v opačném pořadí než dojede k cíli. V tabulce 7 jsou pro představu uvedeny ještě časy výpočtu algoritmu.

5.3 Rozdíl rychlosti dvou fází vyhledávání pomocí metody oddělovačů

Vyhledávání pomocí metody oddělovačů popsané v sekci 4.3 je rozděleno do dvou fází. Tabulka 8 ukazuje vztah mezi počtem komponent a časy vyhledávání jednotlivých fází. Pro každý počet komponent bylo provedeno tisíc vyhledávání se stejnou množinou vybraných testovacích cest. Těchto tisíc výsledků bylo zprůměrováno. V tabulce vidíme,

	Čas výpočtu [s]			
	$l_m = 5$ km	$l_m = 10$ km	$l_m = 15$ km	$l_m = 20$ km
$l_s = 10$ km	0,7	1,2	1,8	2,3
$l_s = 20$ km	1,5	2,3	2,9	3,5
$l_s = 30$ km	2,6	3,1	3,4	3,9
$l_s = 40$ km	3,3	4,0	4,3	4,8
$l_s = 50$ km	4,1	5,0	5,2	5,5

Tabulka 7: Rychlost výpočtu obousměrného omezeného vyhledávání.

Počet komponent	Čas výpočtu [ms]		
	Fáze 1	Fáze 2	Celkem
50	191,1	46,2	237,3
100	84,6	63,6	148,2
200	36,5	78,3	114,8
300	21,6	90,5	111,1
400	15,0	94,5	109,5
500	10,4	103,1	113,5
1000	4,8	115,4	120,2
1500	3,1	121,7	124,8

Tabulka 8: Změřené časy obou fází algoritmu vyhledávání pomocí metody oddělovačů.

že pro malý počet komponent je rychlejší druhá fáze vyhledávání a naopak pro velký počet komponent je rychlejší první fáze vyhledávání. Dle celkového času můžeme usuzovat, že ideální počet komponent je blízko u čísla 400. Tento výsledek musíme však brát s rezervou, protože se vztahuje ke konkrétnímu grafu a jeho rozdělení, které bylo provedeno určitým algoritmem s určitým nastavením. Obecně můžeme pouze říct, že je lepší rozdělení na vyšší počet komponent.

5.4 Oblast výpočtu hraničního grafu a vliv na vyhledávání

Algoritmus vyhledávání s metodou oddělovačů je závislý na předem vytvořeném hraničním grafu. Dříve jsme konstatovali, že musíme provést výpočet nejkratších cest hraničního grafu ve větších oblastech, než pouze v jednotlivých komponentách. Nejjednodušší způsob takového omezení je vytvořit kolem komponent geografický ohraničující obdélník[10]. Dijkstrův algoritmus bude vyhledávat nejkratší cesty pouze v tomto obdélníku. Určíme číslo x, kterým vynásobíme výšku a šířku obdélníku a o tuto vynásobenou výšku, respektive šířku, zvětšíme obdélník do všech směrů. Číslo x je společné pro výpočet celého hraničního grafu. V tabulce 9 jsou uvedeny časy výpočtu nejkratších cest

[10]Toto řešení nemusí být všude stejně přesné, protože Země je kulatá. Komponenty jsou ale velice malé a můžeme na ně tedy nahlížet jako na plošné útvary.

x	Počet nalezených nejkratších cest	Čas výpočtu [s]
0,00	180 384	44
0,25	207 795	143
0,50	209 521	216
1,00	209 946	339
2,00	210 024	530

Tabulka 9: Čas vytváření hraničního grafu pro Českou republiku a počet nalezených nejkratších cest.

x	Čas vyhledávání [ms]	Prodloužení času cesty [%]
0,00	190	1,05
0,25	197	0,93
0,50	198	0,92
1,00	199	0,75
2,00	201	0,69

Tabulka 10: Vliv oblasti výpočtu nejkratších cest hraničního grafu na vyhledávání.

hraničního grafu a počty nalezených nejkratších cest v závislosti na x. Čím větší je x, tím více nejkratších cest nalezneme, ale také tím déle výpočet trvá.

Tabulka 10 ukazuje, jaký vliv má velikost ohraničujícího obdélníku na vyhledávání s metodou oddělovačů. Číslo x jsme již popsali výše. U hodnoty 0 byl výpočet nejkratších cest hraničního grafu proveden pouze v rámci komponent. Hodnoty vyšší než 1 značí, že byly použity ohraničující obdélníky větší, než je velikost komponent. Bylo provedeno vyhledávání metodou oddělovačů se 100 předem vybranými dvojicemi vrcholů. Čas vyhledávání je zprůměrován ze všech výsledků. Prodloužení času cesty je také zprůměrováno a je bráno ve vztahu ke skutečné nejkratší cestě vypočítané Dijkstrovým algoritmem. Z tabulky je patrné, že jsou na sobě jednotlivé hodnoty závislé. Vyšší x znamená, že jsme při vytváření hraničního grafu nalezli více nejkratších cest, a tím se lehce zvýší čas vyhledávání a sníží procentuální odchylka přesnosti od skutečné nejkratší cesty. Můžeme říct, že kdyby nebyla oblast výpočtu nejkratších cest hraničního grafu omezena, nalezli bychom pravděpodobně skutečnou nejkratší cestu.

5.5 Srovnání směrovacích algoritmů

V této sekci provedeme finální srovnání testovaných algoritmů. Seznam testovaných algoritmů i s jejich zkráceným názvem je v tabulce 11.

Pro srovnání jsme museli určit parametry testovaných algoritmů. Všechny testované algoritmy založené na A* algoritmu (konkrétně A*, OA* a OOA*) měly heuristický parametr, kterým je cestovní rychlost, nastaven na 130 kmh^{-1}. Nastavením tohoto parametru na maximální povolenou rychlost v celé silniční síti dosáhneme nejlepších výsledků, viz poznámka o optimalitě ze sekce 4.2.1. Algoritmus obousměrného hierarchic-

Zkratka	Popis algoritmu
DA	běžný Dijkstrův algoritmus
ODA	obousměrný Dijkstrův algoritmus
A*	A* algoritmus
OA*	obousměrný A* algoritmus
OHV	obousměrné hierarchické vyhledávání
OODA	obousměrný omezený Dijkstrův algoritmus
OOA*	obousměrný omezený A* algoritmus
VMO	vyhledávání s metodou oddělovačů

Tabulka 11: Popis testovaných algoritmů

Algoritmus	Čas vyhledávání [ms]	Prodloužení času cesty [%]
DA	10 244	0,000
ODA	6 869	0,000
A*	7 185	0,243
OA*	4 259	0,432
OHV	2 236	2,427
OOD	5 939	0,703
OOA*	2 882	1,159
VMO	197	0,696

Tabulka 12: Srovnání směrovacích algoritmů.

kého vyhledávání měl rychlostní graf vypočítaný na velikost 15 km, což je mnohem více, než jsme dříve používali, a to proto, že je tento algoritmus závislý na kvalitě mapových podkladů stejně jako z testů vyřazené vyhledávání omezené silničními kategoriemi. Obě varianty obousměrného omezeného vyhledávání (OODA a OOA*) měly parametry nastaveny tak, aby nalezly všechny hledané cesty, což znamená $length_{start} = 50$ km a $length_{max} = 25$ km. U vyhledávání s metodou oddělovačů byl graf rozdělen na 400 částí a hraniční graf byl vytvořen s parametrem $x = 2$.

Tabulka 12 ukazuje finální srovnání testovaných algoritmů. Každý algoritmus provedl sto krát vyhledání cesty s předem určenou množinou testovacích dvojic vrcholů. Výsledky algoritmů jsou zprůměrovány ze všech vyhledávání. Prodloužení času cesty je procentuální vyjádření určující zhoršení kvality nalezené cesty vzhledem k Dijkstrovu algoritmu. Vyhledávání s metodou oddělovačů je dle času vyhledání zdaleka nejrychlejší algoritmus. Druhý nejrychlejší algoritmus, kterým je OHV, je v porovnání s VMO více než deset krát pomalejší. Z hlediska prodloužení času cesty jsou všechny algoritmy relativně velice přesné. Nejhorší přesnost vyhledávaných cest má OHV, jelikož byl tento algoritmus navržen s jinými, mnohem kvalitnějšími mapovými podklady. Náš moderní směrovací algoritmus, VMO, je v přesnosti vyhledávaných cest průměrně o 0,7% horší, než DA. Kombinace rychlosti a přesnosti činí z VMO pravděpodobně nejlepší algoritmus tohoto testu. Přesnost by bylo možné ještě dále zlepšovat zvýšením parametru x.

6 DATEX

DATEX představuje rozsáhlý standard pro výměnu dat souvisejících s dopravou. DATEX byl vyvinut pro výměnu informací mezi centry řízení dopravního provozu, informačními centry a poskytovateli služeb. Aktuální verze[11] je DATEX II v2.1.

Hlavní cíl projektu je podpora mobility v Evropě a rozvoj výměny informací mezi aktéry řízení silničního provozu. Výměna informací by měla zajistit zvýšení bezpečnosti na silnicích, plynulejší provoz a v konečném důsledku i zlepšení životního prostředí. DATEX se stal základním kamenem pro všechny aplikace, které vyžadují přístup k dynamickým dopravním informacím v Evropě.

DATEX využívá strukturovaný datový model, který je vytvořen v UML a je platformně nezávislý. Tento model může být implementován mnoha způsoby. Prakticky se však používá pouze jedna implementace, a to pomocí XML. DATEX nedefinuje způsob přenosu dat, a proto si musí poskytovatelé a příjemci určit mezi sebou komunikační kanál. Nejčastěji se pro výměnu dat používají webové služby, http push a pull.

Nejdůležitější část UML modelu DATEXu je abstraktní třída *PayloadPublication*. Tato třída má aktuálně devět potomků, přičemž každý z nich je určen pro reprezentaci jiného typu dat.

Výstup směrovacích algoritmů je velice jednoduchý. Obsahuje pouze základní informace o cestě a geometrii cesty ve formě setříděných GPS bodů. Pro reprezentaci tohoto výstupu v DATEXu se musíme omezit pouze na část UML modelu, která bude pokrývat tyto data.

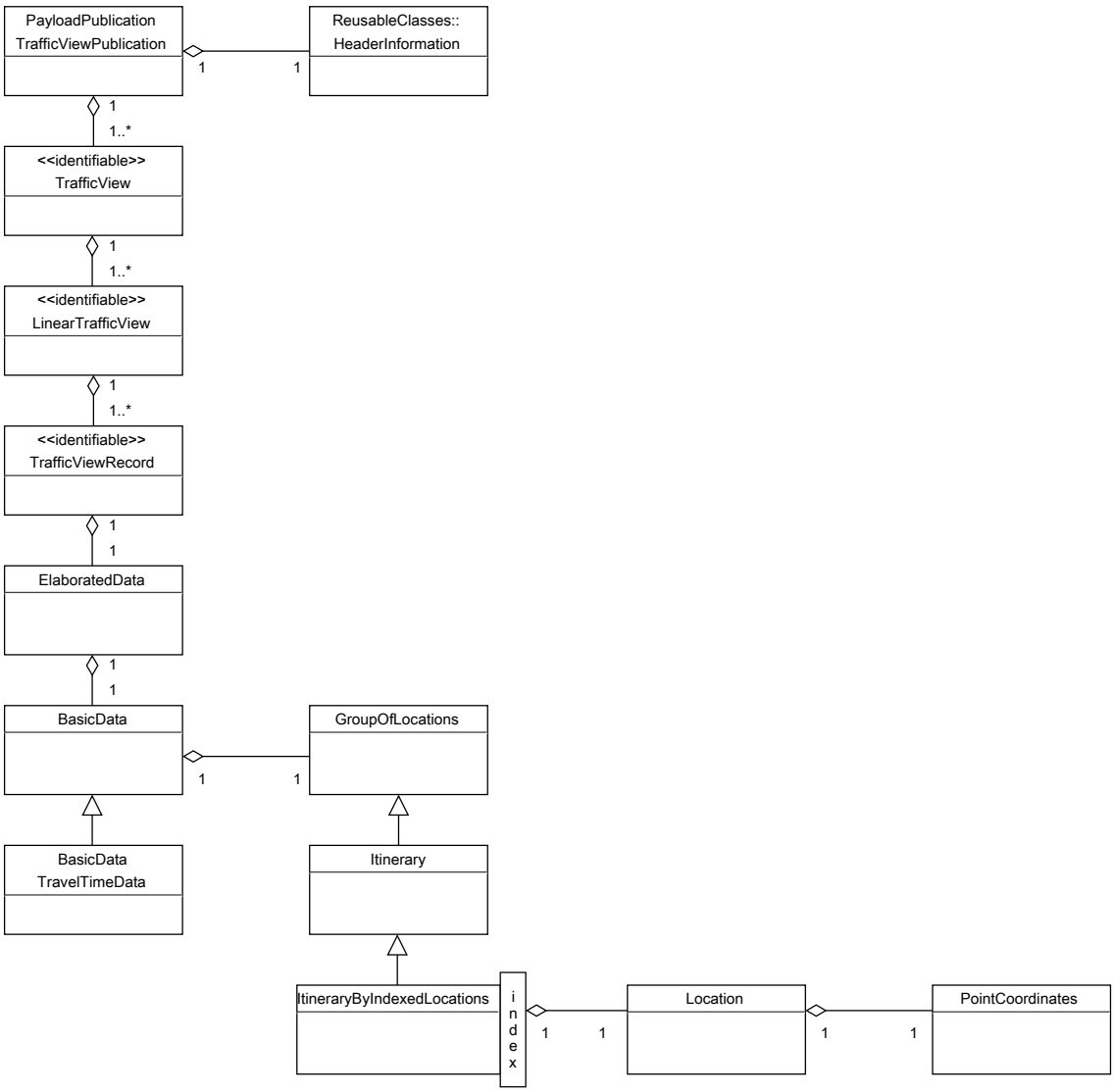
Hlavní prvek našeho modelu bude potomek abstraktní třídy *PayloadPublication* s názvem *TrafficViewPublication*. Na obrázku 15 je zjednodušený UML model použitelný pro reprezentaci výstupu směrovacích algoritmů. Model obsahuje pouze nutné minimum a neobsahuje kvůli čitelnosti atributy. Násobnosti vazeb jsou převážně převzaty z původního modelu a na několika místech jsou přidány povinnosti vztahů. Geometrie cesty je uložena v podobě setříděných bodů ve třídě *ItineraryByIndexedLocations*. Jednotlivé body jsou představovány třídou *PointCoordinates*, která obsahuje zeměpisnou šířku a délku ve formátu WGS 84. Základní informace o dojezdové době je ve třídě *TravelTimeData*.

Náš zjednodušený UML model bude validní dle DATEXu, jakmile do něj doplníme všechny neuvedené povinné atributy. To zde ale již dělat nebudeme, protože je jich opravdu velké množství.

V našem modelu jsme využili nejjednodušší způsob určování polohy pomocí zeměpisných souřadnic. Tento způsob je výhodné použít, pokud se nechceme vázat na prvky silniční sítě. Další možnosti určování polohy jsou *AlertC* a *TPEG*. Tyto možnosti jsou však svázány s použitím lokačních tabulek, a proto pro nás nejsou vhodné.

Hlavní nevýhoda DATEXu je v jeho rozsáhlosti a v nutnosti použít mnoho povinných atributů a tříd, které ve výsledku vůbec nevyužijeme. Výstup směrovacího algoritmu v DATEXu je, co se týká velikosti dat, mnohonásobně větší, než by tomu bylo bez použití tohoto formátu.

[11] Adresa projektu je http://www.datex2.eu/. Na adrese je možné stáhnout všechny dokumenty týkající se DATEXu.



Obrázek 15: Zjednodušený minimální UML model použitelný pro výstup směrování.

7 Závěr

Hlavní cíl této práce byl ve vytvoření moderního směrovacího algoritmu. Pro dosažení tohoto cíle musely být zpracovány mapové podklady a navrženy nové datové struktury. Mapové podklady OSM se ukázaly jako méně kvalitní náhrada dříve používaných mapových podkladů Navteq, a proto nyní nepracují některé směrovací algoritmy stejně dobře, jako dříve. Otestovali jsme nové datové struktury vzhledem ke starým, přičemž nové jsou podstatně rychlejší, než ty staré. Výsledný směrovací algoritmus, pojmenovaný vyhledávání s metodou oddělovačů, se v testech ukázal jako nejrychlejší algoritmus. Přesnost je v porovnání s ostatními algoritmy také dobrá a je možné ji dále zvyšovat změnou parametru x při vytváření hraničního grafu. Dále jsme ukázali, že přidáním dynamických prvků do směrování můžeme reagovat na určité změny silniční sítě. Prakticky jsme využili možnost směrování s dynamickými prvky u výpočtu dostupnosti hasičských stanic se simulovanými záplavami. V závěru této práce jsme navrhli XML model ve formátu DATEX pro předávání výstupu směrovacích algoritmů.

Budoucí rozšíření této práce a hlavně pak našeho moderního směrovacího algoritmu je možné provést několika způsoby. Při vytváření hraničního grafu jsme zjistili, že pro následné rychlé směrování na tomto grafu je nejpodstatnější snížit počet hraničních vrcholů na minimum. Toho docílíme pouze tím, že použijeme lepší algoritmus rozdělující grafy. Další pokračování této práce by se tedy mohlo ubírat směrem implementace vlastního rozdělujícího algoritmu. Během testu dvou fází vyhledávání s metodou oddělovačů jsme si mohli všimnout, že rychlost obou fází je závislá na počtu komponent, na které byl graf rozdělen. Malý počet velkých komponent značí, že je jedna fáze rychlejší a druhá naopak pomalejší. U velkého počtu malých komponent je tomu přesně opačně. Hodilo by se tedy, kdyby bylo možné zkombinovat výhody malých i velkých komponent. Řešení tohoto problému je jednoduché. Velké komponenty se mohou skládat z více malých komponent. Mohli bychom tedy rozdělit graf nejdříve na malý počet velkých komponent a vytvořit s nimi hraniční graf. Následně bychom rozdělili velké komponenty na více malých komponent a opět bychom vytvořili hraniční graf. Dostali bychom dvě úrovně překrývajících se hraničních grafů, se kterými bychom mohli provádět směrování mnohem rychleji, protože by bylo možné využít výhody malých i velkých komponent. Další možný směr rozšíření této práce je v použití dynamických dat při směrování. Můžeme například použít Jednotný systém dopravních informací a směrovat na základě aktuální dopravní situace.

8 Reference

[1] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte. Min-max heaps and generalized priority queues. *Communications of the ACM*, 1986.

[2] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[3] Charles-Edmond Bichot and Patrick Siarry. *Graph Partitioning*. ISTE, 2011.

[4] Béla Bollobás. *Modern graph theory*. Springer, 1998.

[5] Steve Coast. OpenStreetMap. http://www.openstreetmap.org/.

[6] Thomas H. Cormen, Stein Clifford, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 2001.

[7] Dennis de Champeaux and Lenie Sint. An improved bidirectional heuristic search algorithm. *ACM*, 24(2):177–191, 1977.

[8] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *ACM*, 32(3):505–536, 1985.

[9] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In *Proceedings of the 10th international conference on Experimental algorithms*, SEA'11, pages 376–387, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] Edsger W. Dijkstra. *A note on two problems in connexion with graphs*. Numerische Mathematik, 1959.

[11] Pavla Drázdilová, Jan Martinovic, and Katerina Slaninová. Spectral clustering: Left-right-oscillate algorithm for detecting communities. In Pechenizkiy and Wojciechowski [21], pages 285–294.

[12] A. Efentakis, D. Pfoser, A. Voisard, and C. Wenk. Exploiting road network properties in efficient shortest-path computation. Technical report, ICSI, 2009.

[13] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. In *In Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science*, pages 232–241, 2001.

[14] I.C.M. Flinsenberg, M.G. van der Horst, J.J. Lukkien, and J.H. Verriet. Creating graph partitions for fast optimum route planning. *WSEAS Transactions on Computers*, 3(3):569, 2004.

[15] Andrew V. Goldberg, Chris Harrelson, Haim Kaplan, and Renato F. Werneck. Efficient point-to-point shortest path algorithms. web, 2006.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael. *Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. SIGART, 1972.

[17] Petr Havíček. Algoritmy pro směrování dopravních vozidel. Master's thesis, VŠB - Technická univerzita Ostrava, 2010.

[18] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[19] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs, 1979.

[20] Bing Liu. Route finding by using knowledge about the road network. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 27(4):436–448, 1997.

[21] Mykola Pechenizkiy and Marek Wojciechowski, editors. *New Trends in Databases and Information Systems, Workshop Proceedings of the 16th East European Conference, ADBIS 2012, Pozna, Poland, September 17-21, 2012*, volume 185 of *Advances in Intelligent Systems and Computing*. Springer, 2013.

[22] Ira Pohl. Bi-directional search. *Machine Intelligence*, 1971.

[23] Radek Tomis. Optimalizace směrování dopravních vozidel. Bachelor's thesis, VŠB - Technická univerzita Ostrava, 2011.

[24] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 1976.